NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

A SIMULATION OF THE I3 TO D REPAIR PROCESS AND SPARING OF THE F414-GE-400 JET AIRCRAFT ENGINE

by

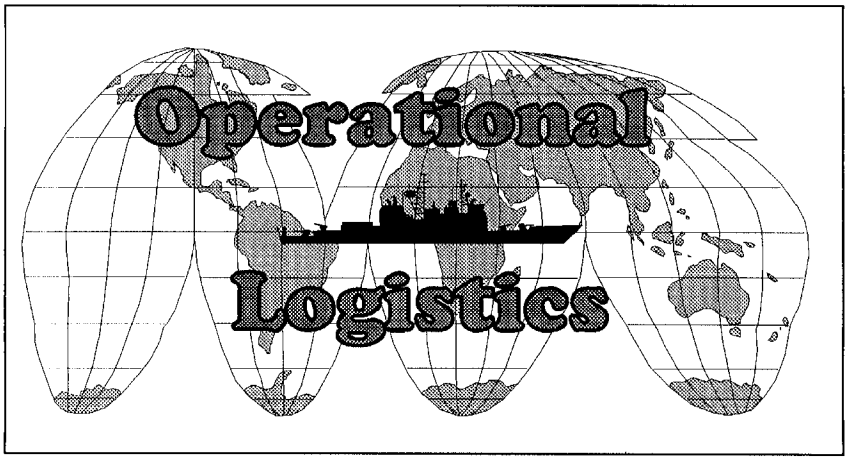
Eric J. Schoch

September 2003

Thesis Advisor:	Arnold H. Buss
Second Reader:	Kevin J. Maher

Approved for public release; distribution is unlimited.

Amateurs discuss strategy, Professionals study logistics



REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: A Simulation of the I3 to D Repair Process and Sparing of the F414-GE-400 Jet Aircraft Engine			**5. FUNDING NUMBERS**
6. AUTHOR(S) Eric J. Schoch			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The F/A-18E/F is the latest multi-mission tactical aircraft to enter United States Naval Service. It generates power via two F414-GE-400 engines, each of which is composed of six modules. In addition to a new aircraft model and engines, a new concept, the I3 to D Repair Process, is being used for F414-GE-400 module and engine repair. In the I3 to D Repair Process, the intermediate level no longer repairs modules. Instead, the depot level performs all module repairs. This thesis develops and exercises a simulation of the I3 to D Repair Process for the F414-GE-400 by incorporating simulated F/A-18E/F flight schedules and engine failures to populate the repair cycle. The simulation provides operational availability (A_0) and probability to spare the repair process given an infrastructure and sparing profile. Three previous years of module failures and depot repair times are used to calibrate the model. Simulation results for the baseline studied showed the distinct influence of certain input parameters. Aircraft service entry time had only a relative short-term effect on A_0. Cannibalization of engines among F/A-18's improved A_0. Scheduled maintenance dramatically impacted A_0. Finally, of all the components of depot repair turn around time (RTAT), "In Work" and "Other" influenced A_0 the most. The simulation was also used to examine the impact of varying build windows and depot RTAT. It allows easy changes of input parameters to be made so that a multitude of effects on A_0 and probability to spare the repair process can readily be studied.

14. SUBJECT TERMS F/A-18, Hornet, F414-GE-400, Jet Aircraft Engine, Simulation, Simkit, Operational Availability, Repair Process, I3 to D	**15. NUMBER OF PAGES** 171
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

A SIMULATION OF THE I3 TO D REPAIR PROCESS AND SPARING OF THE F414-GE-400 JET AIRCRAFT ENGINE

Eric J. Schoch
Lieutenant Commander, Supply Corps, United States Navy
B.S., Iowa State University, 1991

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 2003

Author: Eric J. Schoch

Approved by: Arnold H. Buss
Thesis Advisor

Kevin J. Maher
Second Reader

James N. Eagle
Chairman, Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The F/A-18E/F is the latest multi-mission tactical aircraft to enter United States Naval Service. It generates power via two F414-GE-400 engines, each of which is composed of six modules. In addition to a new aircraft model and engines, a new concept, the I3 to D Repair Process, is being used for F414-GE-400 module and engine repair. In the I3 to D Repair Process, the intermediate level no longer repairs modules. Instead, the depot level performs all module repairs. This thesis develops and exercises a simulation of the I3 to D Repair Process for the F414-GE-400 by incorporating simulated F/A-18E/F flight schedules and engine failures to populate the repair cycle. The simulation provides operational availability (A_0) and probability to spare the repair process given an infrastructure and sparing profile. Three previous years of module failures and depot repair times are used to calibrate the model. Simulation results for the baseline studied showed the distinct influence of certain input parameters. Aircraft service entry time had only a relative short-term effect on A_0. Cannibalization of engines among F/A-18's improved A_0. Scheduled maintenance dramatically impacted A_0. Finally, of all the components of depot repair turn around time (RTAT), "In Work" and "Other" influenced A_0 the most. The simulation was also used to examine the impact of varying build windows and depot RTAT. It allows easy changes of input parameters to be made so that a multitude of effects on A_0 and probability to spare the repair process can readily be studied.

THIS PAGE INTENTIONALLY LEFT BLANK

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs and data herein are free of computational, logic, and collection errors, they cannot be considered validated. Any application of these programs or data without additional verification is at the risk of the user.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

A_0	Operational Availability
AEMS	Aircraft Engine Maintenance System
AIMD	Aircraft Intermediate Maintenance Department
CHINFO	Chief of Information, US Navy Office of Information
CONUS	Continental United States
D-Level	Depot Level of Repair
DES	Discrete Event Simulation
DoD	Department of Defense
DSS	Decision Support System
ELCF	Equivalent Low Cycle Fatigue
EOT	Engine Operating Time
FIFO	First In First Out
HPT	High Pressure Turbine
I-Level	Intermediate Level of Repair
LPT	Low Pressure Turbine
MC	Mission Capable
NALDA	Naval Aviation Logistics Data Analysis
NAVAIR	Naval Air Systems Command
NAVICP-P	Naval Inventory Control Point Philadelphia
O-Level	Organizational Level of Repair
OPNAV	Office of the Chief of Naval Operations

P_{spare}	Probability to Spare the Repair Process
PC ARROWs	Personal Computer (PC) Aviation Retail Requirements Oriented to Weapons Replaceable Assemblies
RFI	Ready For Issue
RIMAIR	Retail Inventory Model for Aviation
RTAT	Repair Turn Around Time
TBF	Time Between Failure
UIC	Unit Identification Code

ACKNOWLEDGMENTS

The author of this thesis would like to acknowledge the contributions of the following people:

To my wife and children. Your love, support, patients, and understanding are beyond measure.

To Dr. Arnold Buss, the creator of Simkit and my advisor. This thesis would have been exponentially more difficult without your insight, experience, help, and guidance.

To CDR Kevin Maher, a fellow Supply Corps Officer and my second reader. Thank you for being persistent.

To Dr. Robert Dell. Thank you for setting up my experience tour.

To Dr. David Olwell. Thank you for your guidance and time spent answering reliability related questions.

To Dr. David Schrady. Thank you for the time spent honing my thesis presentation.

To MAJ Richard Williamson, USMC. Thank you for the time and effort you have invested in this thesis. The countless e-mails, data files, and questions answered are greatly appreciated.

To Mr. Gene Woodburn. Thank you for all of the engine data and time spent explaining it.

To CDR Michael Ropiak. Thank you for your efforts in setting up my experience tour.

To my classmates. It has been an honor and pleasure to work with you these past two years.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The F/A-18E/F is the latest multi-mission tactical aircraft to enter United States Naval Service. It is the planned replacement for the F/A-18A/B, F/A-18C/D, and F-14. The F/A-18E/F generates power via two F414-GE-400 engines that are each composed of six primary modules: the fan, compressor, combustor, high pressure turbine (HPT), low pressure turbine (LPT), and afterburner.

The United States Navy uses three levels of maintenance for aircraft engine repair: the organizational level (O-Level), intermediate level (I-Level), and depot level (D-Level). The O-Level is the operational echelon whose sphere of responsibility includes engine troubleshooting, removal and installation of the engine from the aircraft, and basic engine maintenance. The I-Level, or Aircraft Intermediate Maintenance Department (AIMD), has the capability to perform most in-depth maintenance and repair. The D-Level is the top echelon of the jet aircraft engine repair process and can perform all maintenance and repair actions. Naval Aviation Depot, Jacksonville is the only D-Level that repairs the F414-GE-400 and F404 family of engines.

In addition to a new aircraft model and engines, the United States Navy is using a new concept, the I3 to D Repair Process, for F414-GE-400 module and engine repair. The major difference between the traditional and the I3 to D Repair Process is that the I-Level does not repair modules. Instead, the I-Level sends all modules requiring repair to the D-Level.

This thesis utilizes a simulation of the I3 to D Repair Process for the F414-GE-400. It incorporates simulated F/A-18E/F flight schedules and engine failures to populate the repair process. The goal of the simulation is to provide operation availability (A_0) and probability to spare the repair process given an infrastructure and sparing profile. To do this, it uses three previous years of module failures and depot repair times from the F404 family of engines to calibrate the model.

The simulation allows easy changes to input parameters so that effects on A_0 and probability to spare the repair process can be readily studied. Analysis of results after exercising the simulation over a wide range of changes to the input parameters provides

some useful insights. First, aircraft service entry time has a noticeable, but relatively short-term effect on A_0. Second, A_0 is improved by cannibalization of engines between F/A-18's at the organizational level. Third, scheduled maintenance dramatically impacts A_0, and when considered independently, provides an upper bound for it. Fourth, of all the components of depot repair turn around time (RTAT), "In Work" and "Other" influence A_0 the most. Finally, the build window length currently utilized by the United States Navy in the F414-GE-400 is nearly optimal when considering A_0.

The simulation used in this thesis allows for changes to input parameters to easily be made. As new baseline plans for sparing and aircraft population are developed, analysis through simulation can provide valuable insight into the factors that affect A_0 and probability to spare the repair process.

I. INTRODUCTION

A. BACKGROUND

Interest in fixed-wing aircraft applications in the United States Navy can trace it roots back to 25 March 1898, close to three years before the Wright brothers' first flight on 17 December 1903.

> 25 March 1898 Theodore Roosevelt, Assistant Secretary of the Navy, recommended to the Secretary that he appoint two officers "of scientific attainments and practical ability" who, with representatives from the War Department, would examine Professor Samuel P. Langley's flying machine and report upon its practicability and its potentiality for use in war (Grossnick, 1997, 1).

Captain Hutch I. Cone, Chief of the Bureau of Steam Engineering, submitted the first "requisition" for an aircraft on 7 October 1910. On 4 March 1911, funding of $25,000 became available for Naval Aviation (Grossnick, 1997, 3). Once money was made available to purchase aircraft, it is fair to assume that the issue of parts support and repair soon became a topic of interest.

United States Naval Aviation has changed dramatically from its birth in the early twentieth century. Its transformation is a continual process focused on providing a naval air fighting force second to none. A prime example of this change is evident every time a new model of aircraft is placed into service.

The latest jet aircraft model to enter United States Naval Service is the F/A-18E/F Super Hornet. Based on the F/A-18A/B/C/D Hornet, new technologies and design improvements have advanced the performance of the Super Hornet to the point where it can replace both the Hornet and F-14 Tomcat onboard aircraft carriers. Section 1 of this chapter discusses this topic in greater detail.

With the introduction of every new model of aircraft in the United States Navy, parts support and sustainability are paramount issues of concern. This thesis focuses on the F/A-18E/F Super Hornet, its F414-GE-400 engines, and the repair process flow and supply sparing for the major modules of the F414.

1. F/A-18E/F

The F/A-18E/F Super Hornet is the latest multi-mission tactical aircraft to enter service in the United States Navy. Variants currently in production include the F/A-18E, a single-seat version, and the F/A-18F, a two-seat model. The Super Hornet is the planned replacement for the less capable F/A-18A/B/C/D Hornet and the aging F-14 Tomcat. Table 1 provides general characteristics for comparison of the F/A-18E/F, F/A-18C/D, and F-14.

Characteristic	F/A-18E/F Super Hornet	F/A-18C/D Hornet	F-14 Tomcat
Primary Function	multi-role attack and fighter aircraft	multi-role attack and fighter aircraft	strike fighter
Primary Contractor	McDonnell Douglas	McDonnell Douglas	Northrop Grumman
Propulsion	two F414-GE-400 turbofan engines	two F404-GE-402 enhanced performance turbofan engines	F-14A: two TF30-414A afterburning turbofans; F-14B/D: two F110-GE-400 afterburning turbofans
Thrust	22,000 pounds static thrust per engine	17,700 pounds static thrust per engine	TF30-414A: 20,000+ pounds thrust per engine; F110-GE-400: 27,000+ pounds thrust per engine
Length	60.3 feet	56 feet	62.8 feet
Height	16 feet	15.3 feet	16 feet
Wingspan	44.9 feet	40.2 feet	64 feet unswept; 38 feet swept
Maximum Gross Take Off Weight	66,000 pounds	51,900 pounds	not listed
Range (Combat)	1,275 nautical miles, clean plus two AIM-9s	1,089 nautical miles, clean plus two AIM-9s	not listed
Range (Ferry)	1,660 nautical miles, two AIM-9s, three 480 gallon tanks retained	1,546 nautical miles, two AIM-9s, three 330 gallon tanks	1,600 nautical miles
Ceiling	50,000+ feet	50,000+ feet	50,000+ feet
Speed	Mach 1.8+	Mach 1.7+	Mach 2.0+
Crew	E models: one; F models: two	C models: one; D models: two	two
Armament	one M61A1/A2 Vulcan 20mm cannon	one M61A1/A2 Vulcan 20mm cannon	one M61A1/A2 Vulcan 20mm cannon
External Payload	AIM-9 Sidewinder, AIM-7 Sparrow, AIM-120 AMRAAM, Harpoon, Harm, SLAM, SLAM-ER, Maverick Missile, Joint Stand-Off Weapon, Joint Direct Attack Munition, Data Link Pod, Paveway Laser Guided Bomb, and various general purpose bombs, mines, and rockets	AIM-9 Sidewinder, AIM-7 Sparrow, AIM-120 AMRAAM, Harpoon, Harm, SLAM, SLAM-ER, Maverick Missile, Joint Stand-Off Weapon, Joint Direct Attack Munition, and various general purpose bombs, mines, and rockets	AIM-54 Pheonix, AIM-9 Sidewinder, AIM-7 Sparrow, Joint Direct Attack Munition, and air-to-ground precision strike ordnance

Table 1 Navy Fighter Aircraft Comparison (After: CHINFO, 2003)

The F/A-18E/F Super Hornet's history in the United States Navy and basic design can be traced to its predecessor, the Hornet. The F/A-18A/B and F/A-18C/D Hornet

started operational service in 1983 and 1989 respectively and continue to be used today. The concept for an advanced version of the Hornet, the “Hornet 2000”, was announced in January 1988. Table 2 details key milestones of the F/A-18E/F Super Hornet program from this date and illustrates the relative infancy of the program.

Date	Milestone
January 11, 1988	McDonnell Douglas and the US Navy announce that they are studying concepts for an advanced version of the F/A-18 Hornet called the "Hornet 2000."
May, 1995	General Electric delivers the first F414 engines.
September 19, 1995	F/A-18E1 rolls out at a ceremony where Admiral Boorda, Chief of Naval Operations, names the E/F the Super Hornet.
November 29, 1995	F/A-18E1 completes its first flight.
April 1, 1996	F/A-18F completes its first flight.
January 24, 1997	F/A-18F1 completes initial sea trials aboard the USS John C. Stennis.
September 15, 1997	The Super Hornet enters production at Boeing.
August 13, 1998	General Electric delivers the first F414 production engine.
November 6, 1998	F/A-18E6, the first production Super Hornet, makes its first flight.
December 18, 1998	F/A-18E6, the first production Super Hornet, delivered to the US Navy.
January, 1999	VFA-122, the first Super Hornet squadron, "stands up."
November 17, 1999	VFA-122 receives its first seven Super Hornets at NAS Lemoore.
February 15, 2000	Full rate production of the Super Hornet begins.
December 7, 2000	VFA-115, the first operational Super Hornet squadron, receives its first Super Hornet.
September 26, 2001	Boeing delivers the first full rate production Super Hornet to the US Navy.
July 24, 2002	F/A-18E/F's first deploy with VFA-122 aboard USS Abraham Lincoln.
November 6, 2002	F/A-18E's engage in their first combat activity in response to hostile actions over Iraq's southern "no-fly" zone.
March 3, 2003	VFA-14 and VFA-41 deploy aboard USS Nimitz. This is the first time that a ship's complement has included more than one Super Hornet squadron.

Table 2 F/A-18E/F Super Hornet Milestones (After: Boeing, 2003)

2. F414-GE-400 Jet Aircraft Engine

The F/A-18 Super Hornet utilizes two F414-GE-400 engines capable of producing 22,000 pounds of static thrust each. “Its nine-to-one thrust-to-weight ratio is one of the highest of any modern fighter engine” (Boeing, 2003). Based on the F404-GE-400 and F404-GE-402, the F414-GE-400 incorporates a modular design that “allows for rapid assembly/disassembly, which significantly reduces maintenance time” (General Electric, 2003).

The F414-GE-400 is composed of six primary modules. As with the F404 family of engines, the six modules of the F414-GE-400 are the fan, compressor, combustor, high pressure turbine (HPT), low pressure turbine (LPT), and afterburner. Modules of the same module type are interchangeable among F414-GE-400 engines. For example, an

afterburner removed from one F414-GE-400 is installable on another F414-GE-400. Figure 1 illustrates this modular construction.

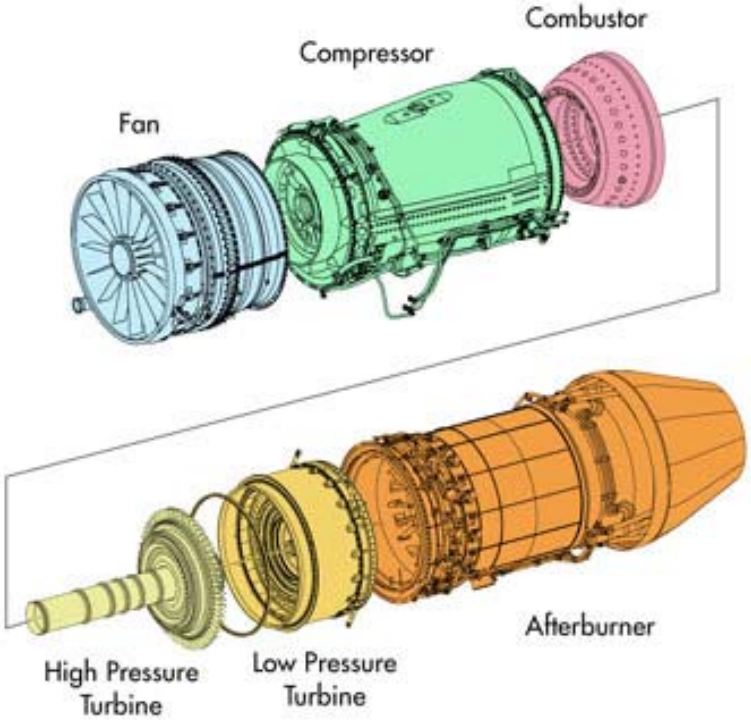


Figure 1 F414-GE-400 Modular Construction (From: General Electric, 2003)

Table 1 illustrates that the F/A-18E/F is a superior aircraft to the F/A-18C/D's. Part of this superiority is obtained from improved engine performance. Table 3 provides general specifications for the F404-GE-400, F404-GE-402, and F414-GE-400 and illustrates how performance has improved while engine size has remained the same.

Engine	F404-GE-400	F404-GE-402	F414-GE-400
Physical Dimensions			
Fan/Compressor Stages	3/7	3/7	3/7
Maximum Length (inches)	154	154	154
Maximum Diameter (inches)	35	35	35
Dry Weight (pounds)	2195	2282	not listed
Application			
Application	F/A-18A/B	F/A-18C/D	F/A-18E/F
Power Specifications			
Maximum Power at Sea Level (pounds)	16000	17700	22000
Overall Pressure Ratio at Maximum Power	26	26	30
Bypass Ratio	0.32	0.29	0.29
Specific Fuel Consumption at Maximum Power	1.85	1.74	not listed

Table 3 F404 and F414 General Specifications (After General Electric, 2003)

Naval Inventory Control Point Philadelphia (NAVICP-P) is responsible for providing program and supply support for the F414-GE-400. The procurement costs for engines and modules play a key role in overall sparing when budgetary constraints are applied. Table 4 lists the engine and module costs that NAVICP-P uses for procurement calculations.

Part	Price $K FY03
Engine	$3,564
AFB Module	$655
CMB Module	$183
FAN Module	$466
HPC Module	$858
HPT Module	$201
LPT Module	$597

Table 4 F414-GE-400 Engine and Module Prices (After: Williamson, 26 August 2003)

3. United States Navy Jet Aircraft Engine Repair

The United States Navy uses three levels of maintenance: the organizational level (O-Level), intermediate level (I-Level), and depot level (D-Level), for repair of jet aircraft engines. The O-Level is the operational echelon whose sphere of responsibility includes engine troubleshooting, removal and installation of the engine from the aircraft, and basic engine maintenance. The I-Level, or Aircraft Intermediate Maintenance Department (AIMD), has the capability to perform most in-depth maintenance and repair. For the F404-GE-400 and F404-GE-402, "the level of repair accomplishable at the intermediate level is almost as extensive as that at the depot level" (Stearns, 1998, 7). Both the O and I-Level can be shore-based or sea-based as a detachment on board an aircraft carrier. The D-Level is the top echelon of the jet aircraft engine repair process and can perform all maintenance and repair actions. Naval Aviation Depot, Jacksonville is the only D-Level that repairs the F414-GE-400 and F404 family of engines.

Figure 2 details the basic flow of a jet aircraft engine through the three levels of repair. Although each level has a "repair engine" block, the level of repair capability varies as previously discussed. The broken engine block in Figure 2 indicates that an engine has experienced a mechanical failure or that the engine and/or one or more modules in the engine have reached "high time," a planned maintenance at predetermined

intervals. When an engine or module in the engine has reached high time, the engine must be removed from the aircraft so that high time maintenance can be performed. Another term related to high time is "build window", a period of time prior to high time in which high time maintenance will be performed on a module if the engine is in repair for any reason.

An example using an automobile illustrates the concept of high time and build window. Assume that the engine on a car has a 3,000 mile high time for oil changes. This means that every 3,000 miles, the car will go in for service and get its oil changed. Now assume that the car's build window is 300 miles. Then, if work is being done on the auto's engine for any reason and it has been between 2,700 and 3,000 miles since the last oil change, the oil will be changed as well.

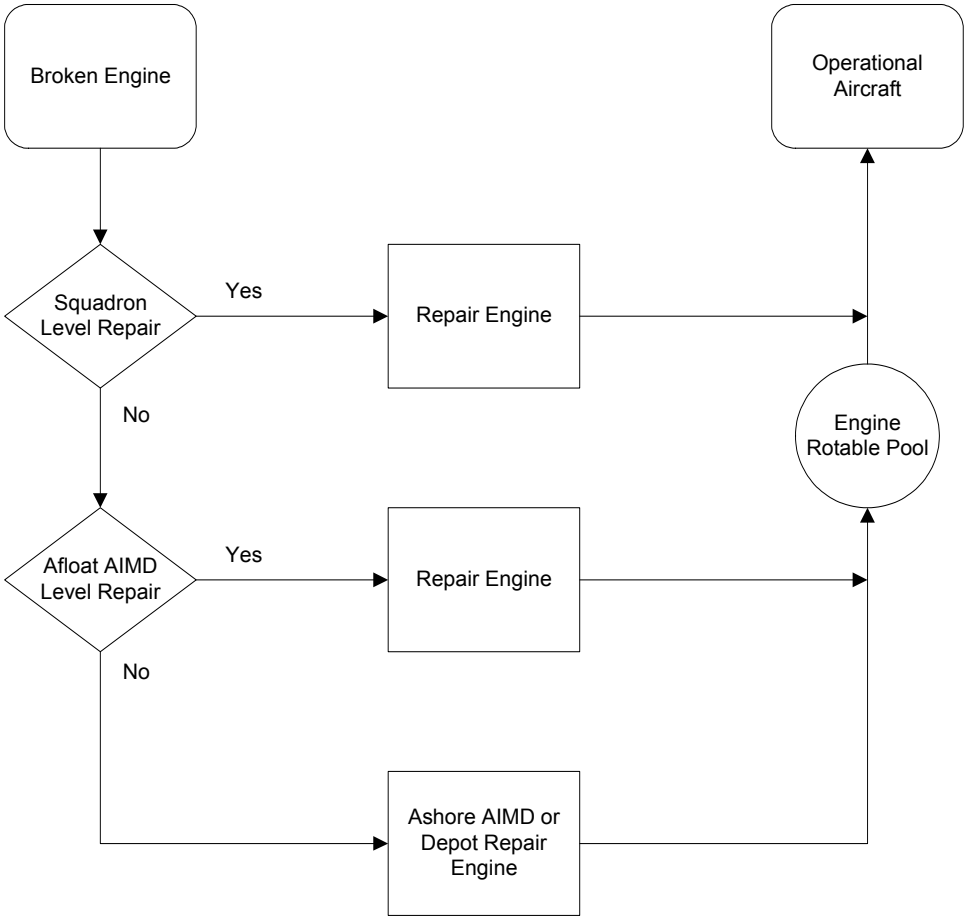


Figure 2 Jet Aircraft Engine Repair Cycle Overview (From: Stearns, 1998, 14)

The engine rotable pool displayed in Figure 2 represents a physical grouping of ready-for-issue (RFI) jet aircraft engines. When an aircraft experiences an engine failure or high time, an engine from the engine pool is provided as replacement. Thus, an aircraft can be placed back into service much more rapidly than if it would have to wait for repair of its removed engine. Figure 3 illustrates the O-Level portion of the engine repair cycle and this concept.

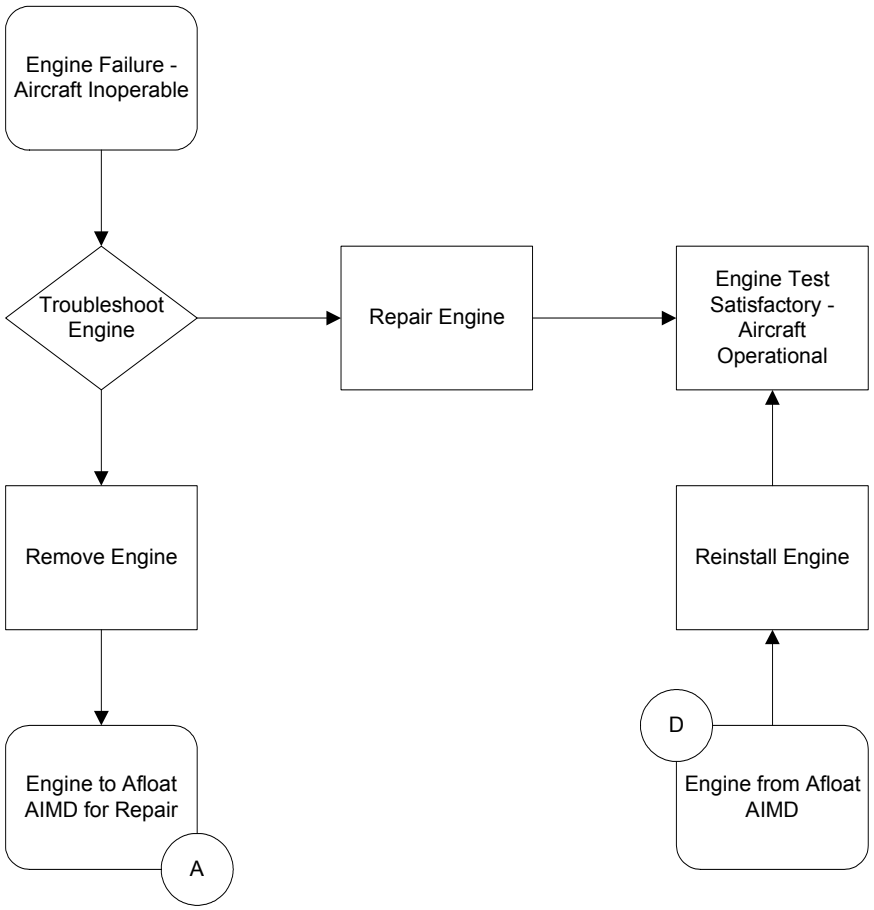


Figure 3 O-Level Jet Aircraft Engine Repair Cycle (From: Stearns, 1998, 15)

A module and engine rotable pool concept is utilized by the I-Level for repair of both the F414 and F404 families of engines. When a F/A-18's failed engine is received at the I-Level, the modules causing the engine failure or needing high time repair can be removed and replaced by modules in the rotable pool or usable modules from other failed engines. Thus, the modular design of these engines allows for quicker repairs than non-modular engines. Figure 4 details the I-Level jet aircraft engine repair process for the

F404-GE-400 and F404-GE-402 engines. A discussion of the F414-GE-400 I3 to D Repair Process, a variant of Figure 4 follows.

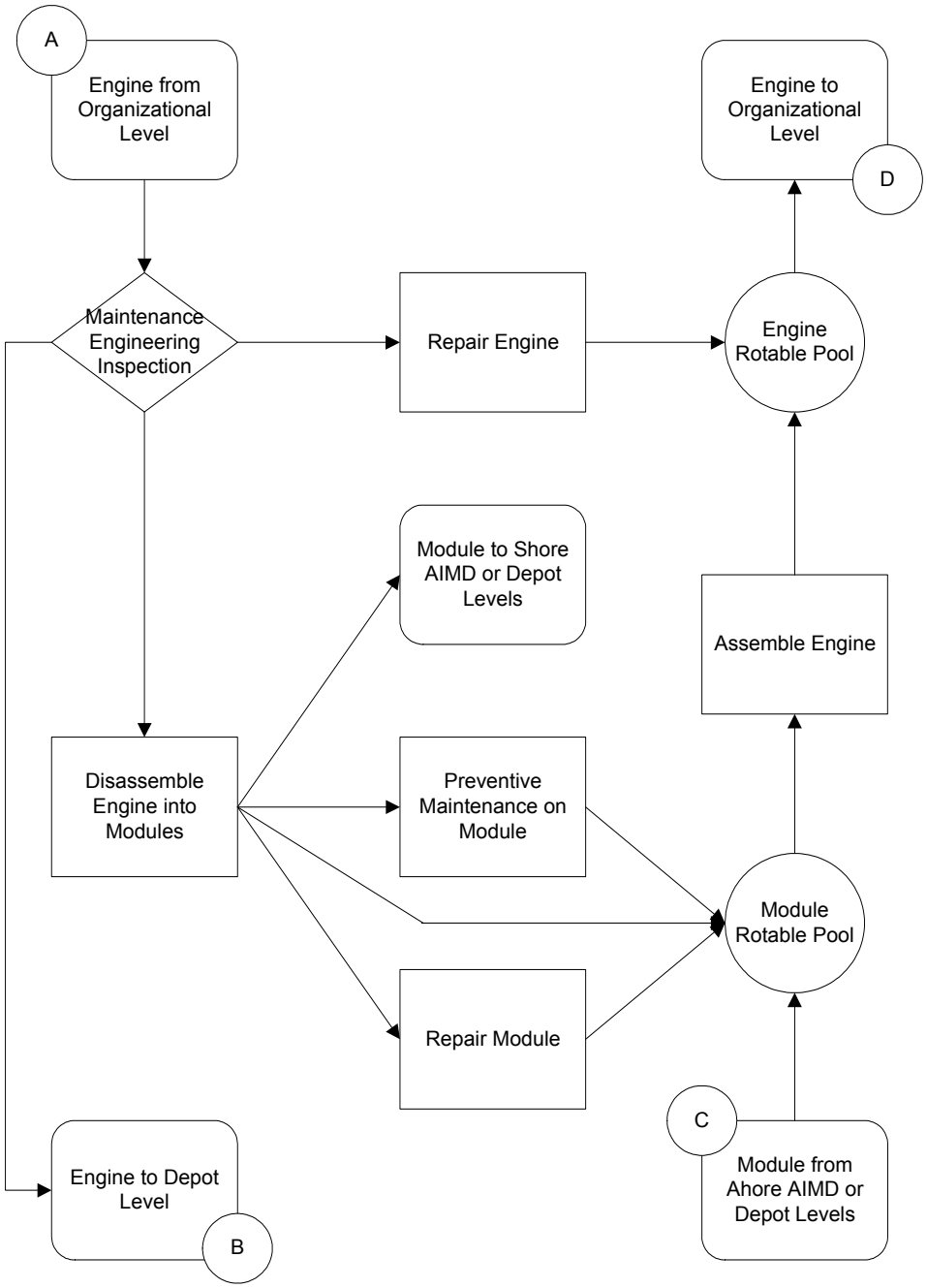


Figure 4 F404 I-Level Jet Aircraft Engine Repair Cycle (From: Stearns, 1998, 17)

The I3 to D Repair Process of the F414-GE-400 is a departure from the traditional method of module repair detailed in Figure 4. In the I3 to D Repair Process, the I-Level still maintains the capability to assemble the F414-GE-400, but no longer repairs failed or

high time modules. Instead, all modules requiring maintenance or repair are shipped to the D-Level. Figure 5 provides an overview of the D-Level repair process.

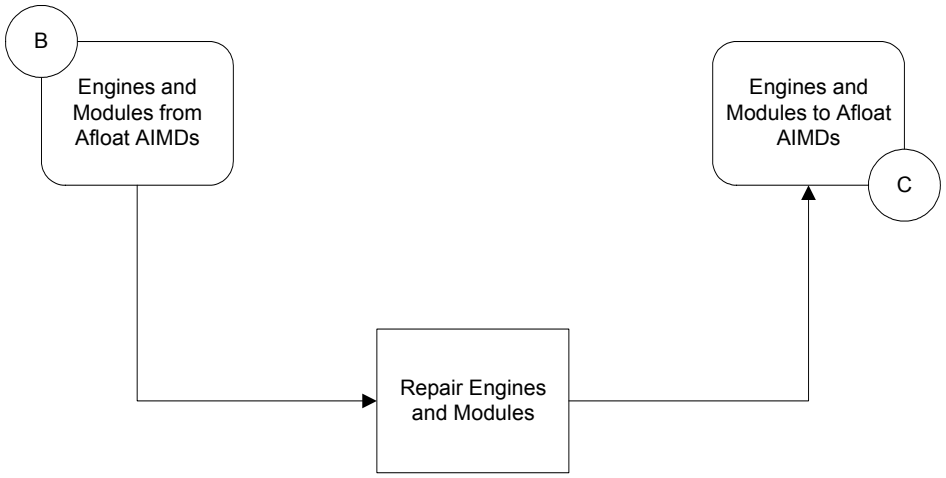


Figure 5 D-Level Jet Aircraft Engine Repair Cycle (From: Stearns, 1998, 18)

Figure 3 thru Figure 5 detail the jet aircraft engine repair process onboard an aircraft carrier. The repair process for the ashore O and I-Levels are nearly identical to those afloat.

4. Operational Availability

The collective supply and repair system goal is to provide and maintain the weapon systems required to win wars. Therefore, the logical measure of effectiveness for this support is the ratio of time that a weapon system is available for warfighting to the lifespan of the system. This ratio, operational availability (A_0), is a readiness indicator.

Operational availability is "a measure of the likelihood that a Weapon System is able to perform the missions for which it was designed at a random point in time" (Burrows, 1994, 2-2). "It is Navy policy that A_0 shall be the primary measure of material readiness for weapon systems and equipment" (OPNAV, 1987, I-2). In basic mathematical terms, operational availability is:

$$A_0 = \frac{time\ weapon\ system\ is\ available\ to\ perform\ mission}{total\ time} \qquad \text{(Equation 1)}$$

The F/A-18 Super Hornet is the sum of numerous systems. For the Super Hornet to be available to perform its mission, it must have two fully operational engines. When

focusing specifically on engine contributions to overall operational availability, Equation 1 can be rewritten as:

$$A_{0F/A-18} = \frac{time\ F/A-18\ had\ two\ fully\ functional\ engines}{total\ time\ for\ F/A-18} \qquad \text{(Equation 2)}$$

Equation 2 represents the operational availability of a F/A-18 given that all systems besides the engines are one hundred percent reliable. It is therefore an upper bound of the actual operational availability.

The operational availability of all Super Hornets for the United States Navy is the sum of the time each F/A-18E/F was available to perform its mission divided by the total amount of time F/A-18E/F's could have flown. This mathematically equates to:

$$A_{0UnitedStates\ Navy} = \frac{\sum_{i=1}^{total\#F/A-18's} time\ F/A-18_i\ had\ two\ fully\ functioning\ engines}{\sum_{i=1}^{total\#F/A-18's} total\ time\ for\ F/A-18_i} \qquad \text{(Equation 3)}$$

B. PURPOSE

NAVICP-P utilizes Personal Computer (PC) Aviation Retail Requirements Oriented to Weapon Replaceable Assemblies (ARROWs) – Retail Inventory Model for Aviation (RIMAIR) to determine sparing requirements for the F414-GE-400 engine and its associated modules.

> PC ARROWs is a budget execution model. You supply the budget figure or the MC/FMC rate or SSR [sortie success rate]. It then determines the range and depth of spares to be stocked for the maintenance and operation of aviation weapons and equipments. The model can determine both provisioning requirements and replenishment requirements (e.g., CVs, Navy Depots (NADEPS) and Naval Air Stations (NASs)) (Burrows, 1994, 1-1).

PC ARROWs is a steady state model that relies "heavily on mean pipelines, expected backorders, and average wait times" (Burrows, 1994, 2-5).

With only a steady state model, NAVICP-P cannot study the effects of various input parameters on operational availability over the lifetime of the program under

consideration. For example, PC ARROWS – RIMAIR cannot provide projected operational availability at, say, the five-year point (assuming the program is not at steady state after five years). Additionally, one cannot readily study the effects that the components of the inputs have. High time is a perfect illustration of this point. An input that NAVICP-P uses in its PC ARROWS - RIMAIR model is mean times between removals for modules. High time removals contribute to the times between removals input, but cannot be separated from them. Therefore, there is currently no way to study the effects of high time removals on operational availability.

This thesis developed a simulation that provides estimated operational availability and the probability to spare the repair process (two measures of effectiveness for the supply support provided) given a base set of input parameters (a baseline) and any changes to those parameters. The probability to spare the repair process (P_{spare}) is the probability that at any time during the period under consideration, a replacement engine is available for an aircraft that needs ones. This is also referred to as the probability of "no bare firewalls." A firewall is the physical barrier that separates the engine from the rest of the aircraft. Thus, "no bare firewalls" indicates that every aircraft engine compartment has either an engine installed or one available for installation.

The purpose of this thesis is to provide NAVICP-P a non-deterministic means to explore the effects that changing various input parameters have on operational availability and P_{spare} in the I3 to D Repair Process taking into account the randomness inherent in the system. In essence, the purpose is to create a management and decision making tool. The simulation is not exercised to generate a sparing policy (although this is a possible area for future research discussed in Chapter V). Instead, the simulation determines operational availability and P_{spare} over time for a series of "what if" questions. The simulation takes into account NAVICP-P provided engine/module sparing plan, aircraft population, flight hours, and repair infrastructure from the O to D Level. A sampling of questions that can be answered include the following. A detailed description of the questions asked and their results can be found in Chapter V.

- "What if" the mean repair turn around time was reduced at the depot?
- "What if" modules were more reliable, i.e., increased MTBF?
- "What if" the length of build window was changed?

C. METHODOLOGY

The methodology used in this thesis is to develop a discrete event simulation of the I3 to D Repair Process.

> Discrete-event simulation concerns the modeling of a system as it evolves over time by a representation in which the state variables change instantaneously at separate points in time. (In more mathematical terms, we might say that the system can change at only a countable number of points in time.) These points in time are the ones at which an event occurs, where an event is defined as an instantaneous occurrence that may change the state of the system (Law, 2000, 6).

The results obtained from a set experimental design undergo analysis to determine how changes between specific experiments affect output.

> When we perform a simulation study, we would like to know what input factors (decision variables) have the greatest impact on performance measures of interest. Experimental designs tell us what simulation experiments (runs) to make so that the effect of each factor can be determined. Some designs allow us to determine interactions among factors (Law, 2000, 213).

For this thesis, operational availability and the probability to effectively spare the repair process are the performance measures of interest. This is in line with Department of Defense guidance that states "for items that are essential to weapon system performance, the inventory performance goals shall relate to the readiness goal of the weapon system throughout its life cycle, e.g., operational availability" (DoD, 2003, C2.4.2.2.1).

To obtain the performance measures, NAVICP-P stipulated that randomness be incorporated throughout and based upon actual engine/module failure and depot repair times for the F404 family of engines (Williamson, 9 December 2002). This thesis utilized a simulation to model the complex nature of the problem and meet NAVICP's requirements. The extensive amount of historical data for the F404 family (the F414 does not have sufficient data available since the program is in its infancy) allowed the randomness to be incorporated through random draws with replacement from actual observations, thus negating the need to approximate through fitting of distributions. Only simulation provides the capability to use the observations in such a manner.

D. ORGANIZATION OF STUDY

Chapter II details the sources of data and how the data was collected and organized. Chapter III provides an in-depth discussion on how the model and simulation were developed to include assumptions made and experimental design. Chapter VI focuses on results and analysis of the results. Chapter V provides conclusions and further recommendations. The appendices provide supporting information for the chapters.

THIS PAGE INTENTIONALLY LEFT BLANK

II. OVERVIEW OF THE DATA

The purpose of Chapter II is to detail the sources of data used in this thesis, how the data was collected, and if required, how the data was refined for implementation in the simulation.

A. DATA RESOURCES

All data used to provide randomness to the simulation utilized in this thesis, specifically removal and depot repair time histories for each type of module, was obtained from the Naval Air Systems Command (NAVAIR). Section B of this chapter contains a detailed discussion on the method of collection and refinement.

Values for O and I-Level module and engine removal, installation, and inspection times were obtained from the mean values of statistical distributions in a previous thesis on F404-GE-400 engine maintenance at an afloat AIMD (Stearns, 1998, 24-25). For the purpose of this thesis, it is assumed that these distributions are valid for the F414-GE-400 at all afloat and shore I-Levels. Stearns' thesis classifies them as "insignificant parameters for the simulation runs" (Stearns, 1998, 24). The above assumption and use of distribution means therefore have minimal impact on overall outcome. The times were included to add more "realism" without dramatically increasing the complexity of the simulation.

NAVICP-P provided all other data required to complete this thesis. In many instances, data assumed the form of program projections and assumptions. Appendix A details the planned level of sparing, aircraft, and repair infrastructure used in this thesis. Table 7 of Appendix A is the projected number, by activity, of Super Hornets that the United States Navy intends to have in its inventory for each fiscal year. Table 8 is the engine and module sparing by activity for these aircraft based on the PC ARROWS – RIMAIR input of Table 9. Table 7, 8, and 9 make up input referred to as Baseline1 for the purpose of this thesis.

NAVICP-P provided module high times in the form of a F414-GE-400 Periodic Maintenance Information Card Deck Data. Development of the card deck data into

parameters usable by the simulation is discussed in Section B of this chapter. Detailed assumptions and the values used for these assumptions are discussed in Chapter III (Simulation Model Development).

B. DATA COLLECTION AND REFINEMENT

1. Module Time Between Failures

Raw data was obtained from NAVAIR's Naval Aviation Logistics Data Analysis (NALDA) Aircraft Engine Maintenance System (AEMS) Decision Support System (DSS) to develop module times between failures. Since the F414-GE-400 is a relatively new engine and does not have an extensive history of module failures, data from the F404-GE-402 is used instead. Chapter III (Simulation Model Development) discusses this assumption.

An AEMS DSS Time/Flight Hour Usage by Serial Number (TMSREMSERNO) report was generated for each module type of the F404-GE-402. The TMSREMSERNO report lists by module serial number the usage time on the module when it was removed, the removal reason code, and the date when removed. All module failure data used in this thesis were taken from TMSREMSERNO reports generated in Microsoft Excel® spreadsheet format (AEMS DSS allows data output via the Internet in either html or Excel® formats) by the author on 03 March 03.

The TMSREMSERNO report lists all module removals regardless of the reason for the removal. The removal reason code indicates why the module was removed. Appendix B gives a detailed listing of these codes. To obtain actual times between failures, all non-failure related module removals (such as cannibalizations and high times) were deleted from the spreadsheets. Some specific codes not related to module failure and thus removed include 1Y, 3E, 3W, 3X, 4A, 4D, 4E, 4L, 4W, 5G, 6V, 7C, 7D, 7E, and 7G (Woodburn, 11 June 2003). After the module spreadsheets were scrubbed of non-failure related removals, each specific module (as identified by its serial number) was examined. Times between failures (based on module usage in hours and not calendar time) were calculated for each specific module as the difference between its

successive times at removal. Once refined, the times between failures for each module type were converted into a text format file utilized by the simulation.

2. Depot Repair Turn Around Times for Modules

F404-GE-400 depot repair turn around times (RTAT) for the last three years were compiled from AEMS DDS by the AEMS Database Administrator and forwarded to the author (Woodburn, 13 August 2003). The approximation of using F404-GE-400 repair turn around times is necessitated by the lack of data for the F414-GE-400 (as was the case with removal times). In addition, the F404-GE-400 depot RTAT database had dramatically more observations that that of the F404-GE-402. Turn around times for each module type consisted of four distinct periods: Awaiting Parts (AWP), Awaiting Maintenance (AWM), In Work, and Other times. The data for each distinctive period was converted into a text format file used by the simulation. Thus, twenty-four separate data files were created (6 module types x 4 distinct periods of repair).

3. Module High Times

Module high times were derived from the F414-GE-400 Periodic Maintenance Information Card Deck Data provided by NAVICP-P (Williamson, 30 April 2003). The card lists maintenance intervals (high times) by either engine operating time (EOT – the actual time that the engine or module was used) or equivalent low cycle fatigue (ELCF – stress due to changing loads). Since the simulation is time based, all ELCF numbers required conversion into their EOT equivalents. Subject matter experts at NAVICP-P determined a conversion factor of 1.2 ELCF to EOT (Williamson, 25 August 2003). Appendix C lists F414-GE-400 module high times by EOT.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SIMULATION MODEL DEVELOPMENT

A. INTRODUCTION

A stochastic simulation provides an excellent means for testing how changes in input parameters affect the complex interactions among the system's variables over time. A deterministic model cannot provide this degree of resolution. The simulation used in this thesis was designed to be as accurate a representation of reality as possible. However, certain modeling assumptions were made due to the complex nature of the system being studied.

This chapter details simulation development from assumptions to experimental design. Section B of this chapter discusses the assumptions made in the simulation. Section C provides an overview of the simulation package Simkit and the event graph notation used in Section D. Section D provides a detailed description of each Java Programming Language class used to run the simulation and how it interacts with the other classes in the package. Section E details the experimental design used to obtain the results found in Chapter IV (Results and Analysis).

B. ASSUMPTIONS

The simulation uses the following assumptions:

- All levels of repair in the I3 to D Repair Process operate 24 hours a day.
- A year, for the purpose of this simulation, is exactly 52 weeks long.
- Engines and modules are always repaired and never condemned or replaced by new stock (Stearns, 1998, 22).
- The D-Level repairs all modules (Williamson, 09 December 2002). The I-Level does not.
- The I-Level repairs all engines (Williamson, 09 December 2002). The D-Level does not.
- The I-Level has unlimited engine repair capacity.
- Engine repair at the I-Level begins as soon as there are enough modules to complete an engine.
- The O-Level has unlimited engine removal and installation capacity.

- One can approximate the F414-GE-400 jet aircraft engine module times between failures and D-Level RTAT by using three previous years of data from the F404 (Williamson, 09 December 2002). F404-GE-402 module TBF and F404-GE-400 D-Level RTAT times are used.
- General Electric has not improved the F404 in the last three years or incorporated changes that would affect the distribution of module time between failures (Williamson, 09 December 2002).
- The D-Level has not incorporated improvements or changes to the module repair process that could have affected the distribution of the F404 module RTAT's.
- Engine high times fall on module high times. Although there are engine and module high times in the F414-GE-400 Periodic Maintenance Information Card Deck Data, module high times are only considered (Williamson, 09 December 2002).
- The shipping times between the I and D-Levels are a constant. Shipping times for modules are five days for I-Levels in the Continental United States (CONUS) and 15 days for all other I-Level sites (including aircraft carriers).
- The contribution of variability in I-Level engine repair and module removal/installation times is insignificant in regards to the overall repair process (Stearns, 1998, 24).
- The O-Level removes an engine from a F/A-18 only for module failures and high times. All other maintenance requirements have a negligible effect on overall operational availability.
- F414-GE-400 engine failures are the result of independent failures of the modules in the engine. Dependency among module failures can be modeled by assuming that given some time interval, t, if one module fails in the engine and another is within t hours of failing, that module fails also. In this simulation, t is 20 hours.
- 1.2 ELCF equals 1 EOT for calculations of engine high times (Williamson, 25 August 2003).
- High times for creep hours can be ignored (Williamson, 11 August 2003).
- There is a zero probability of damaging or reducing the reliability of an engine during the engine cannibalization process at the O-Level.
- Once an F/A-18's flight schedule is set, the F/A-18 does not vary from it (except in the case of engine failure or high time in which it is not flyable).
- F/A-18's fly a peacetime flight schedule of 35 hours per month (Williamson, 11 August 2003).
- If a F/A-18 experiences an engine failure, it lands immediately. The model assumes no return to origin delay.

- A F/A-18 may have both engines fail simultaneously. The probability of such an event occurring in the simulation approaches zero. However, the simulation maintains this possibility as a design feature.

C. SIMKIT AND EVENT GRAPH NOTATION

The simulation developed for this thesis utilizes the Java package Simkit.

> Simkit is a software package for implementing Discrete Event Simulation (DES) models. Simkit is written in Java and runs on any operating system with Java 2^{TM} installed (Buss, 2001, 15).

Simkit was developed and is maintained by Professor Arnold Buss of the Naval Postgraduate School in Monterey, California.

Section D of this chapter discusses the Java classes used to run the simulation. Classes that model discrete events will also contain a basic event graph (a graphical representation of the underlying logic) in their description. The basic format for an event graph is provided in Figure 6. Nodes A and B represent events in the simulation. A condition is indicated by (i). The letter *t* indicates the time delay between events A and B if condition (i) is satisfied. Thus, the basic event graph of Figure 6 can be interpreted as: "When event A occurs, then if condition (i) is true, event B is scheduled to occur after a delay of *t* simulated time units" (Buss, 2001, 16).

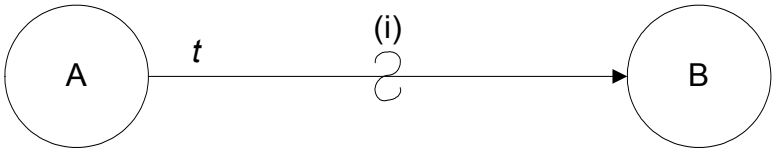


Figure 6 Basic Event Graph Notation (From: Buss, 2001, 16)

Simkit makes it possible to have one object with an event graph "listen" to the event graph of another object in a simulation. When the first object (listener) hears the second object executing an event, the first object executes the matching event if it is contained within its event graph.

D. JAVA CLASSES USED IN THE SIMULATION

The following sections discuss the major design features of the Java classes used to run the simulation. An explanation of a Java class is as follows:

> An object is a program construction that has data (that is, information) associated with it and that perform certain actions. When the program is run, the objects interact with one another in order to accomplish whatever the program is designed to do. The actions performed by objects are called methods. A class is a type or kind of object. All objects in the same class have the same kinds of data and the same methods (Savitch, 2001, 17).

The simulation uses twelve classes (this does not include the classes found in Simkit) whose code is located in Appendices D thru O. Although there is only one of each class, it is possible to instantiate (make a copy using specific input variables) as many objects (copies) from each class as desired. This is the power and utility created in object-oriented programming languages such as Java. For example, the simulation uses the *F18Hornet* class to represent a F/A-18. Since the baseline used in the simulation has a population of 217 F/A-18's, the *F18Hornet* class is instantiated 217 times. The 217 *F18Hornets* created are distinct in that they each retain individual characteristics (such as total hours flown and lifespan), but all perform the exact same methods (actions) of the one *F18Hornet* class (take off and landing for example).

The sections below list by class name the classes, required input, and primary states (that have the most direct impact on the simulation) tracked as they appear in the Java code found in Appendices D thru O. All class names are italicized in the sections below (section titles are not). Comment lines in the Java code describe all states.

1. ModuleType

Appendix D contains the code for the *ModuleType* class. *ModuleType* acts as the sole fixed list of all module types usable by other classes in the simulation. This reduces the possibility of module naming or referencing errors in and among classes.

2. Module

Appendix E contains the code for the *Module* class. The *Module* class is the computer code representation of a physical engine module. Instantiating a *Module* object requires the following input:

- **type –** The type of module from the *ModuleType* class.
- **serno** – The serial number of the module. Although this thesis does not use serno, it is included for future research that could require serial number tracking of modules.
- **rv –** The array of random variates from which times between failures are drawn. If one random variate is in the array, then all random times between failures are drawn from it. If more than one random variate is in the array, the first random time between failure is drawn from the first random variate, the second from the second, and so on. If the number of failures exceeds the number of random variates in the array, the last random variate in the array is used. Results for this thesis are obtained using only one random variate in the array. Although not utilized, the Module class retains this feature for possible future research into module lifecycle changes.

During construction of a *Module* object, a list of module type specific high times is obtained from the *EngineBlueprint* class. *Module* tracks the following primary states:

- **timeOnModule –** The time that the module has been operated.
- **timeBetweenRepair -** The operational time since the module was last repaired. The only two reasons for repair are failure and high time.
- **engineIn -** The engine in which the module is installed in.
- **timeToHighTimes –** An array that tracks the module's high times. Initially, the high time values are set to the *EngineBlueprint* high times for the module's specific type. As the module is used, the values in the timeToHighTimes array are decremented. When a module is repaired, any high time in the array less than the build window is reset to its original value. This simulates accomplishing high time repair(s).
- **timeToFailure** – The operational time until the module fails. When a module fails and is repaired, a new timeToFailure is draw from rv.

3. EngineBlueprint

Appendix F contains the code for *EngineBlueprint* class. *EngineBluprint* is a class that sets the following for each module type in the *ModuleType* class: position and type (see *ModuleType*) of each module in an engine, build window (each module can have a different build window if desired), and failure dependency time (if one module of the engine fails and the other is under the dependency time, it fails also). *EngineBlueprint* sets O-Levels engine removal, installation, and troubleshooting times. It also sets I-Level engine inspection time, engine build times (given that all modules are available to build an engine), and module removal/installation times for each module

type. "Getter methods", methods designed to return a specific variable's content, in *EngineBlueprint* allow all classes to access the above information.

4. Engine

Appendix G contains the code for the *Engine* class. The *Engine* class is the computer code representation of a physical F414-GE-400 jet aircraft engine. Instantiating an *Engine* object requires the following input:

- **serno –** The serial number of the engine. Although this thesis does not use serno, it is included for future research that could require serial number tracking of engines.
- **modules –** An array of modules that make up the engine. When the array of modules is initially passed to create an Engine object, its conformity to the module order and type set in *EngineBlueprint* is checked. Thus, an engine cannot be created that does not have the correct number, type, and order of modules.

Engine tracks the following primary states during a simulation run:

- **timeOnEngine –** The total engine operation time (EOT) – i.e. how many hours the engine has been run in its lifetime.
- **timeBetweenRepair -** The operational time since the engine was last repaired. The only two reasons for repair are failure and high time.
- **timeToHighTime** – The minimum among all installed modules timesToHighTime.
- **timeToFailure** – The minimum among all installed modules timesToFailure.

5. F18Hornet

Appendix H contains the code for the *F18Hornet* class. The *F18Hornet* class is the computer code representation of a physical F/A-18. Instantiating an *F18Hornet* object requires the following input:

- **serno –** The serial number of the F/A-18.
- **engine1 and engine2 –** The two F414-GE-400 used to power the F/A-18. These are from the *Engine* class.
- **flightSchedule –** An array that represents a one week flight schedule for the F/A-18. The flightSchedule is organized into pairs of ground and flight times (example: {gound time 1, flight time 1, ground time 2, flight time 2, ground time 3, flight time 3}). When the F/A-18 reaches the last flight time in its flightSchedule, it starts over at the beginning of the array.
- **weeksDelay –** The number of weeks delay before the aircraft begins executing its flightSchedule for the first time.

When instantiated, the *F18Hornet* obtains and stores its simulation creation time. At any time in a simulation run, the *F18Hornet*'s lifespan is the difference between the current simulation time and creation time. *F18Hornet* tracks the following primary states during a simulation run:

- **flightSchedulePointer –** Where in the flightSchedule the aircraft is at.
- **totalDownTime** – The total time that the aircraft was non-mission capable.
- **totalMissedFlightTime –** The total amount of flight time (mission time) the aircraft missed.
- **totalActualFlightTime –** The total executed amount of flight time.

F18Hornet is most basic element of the simulation that has an associated event graph (Figure 7).

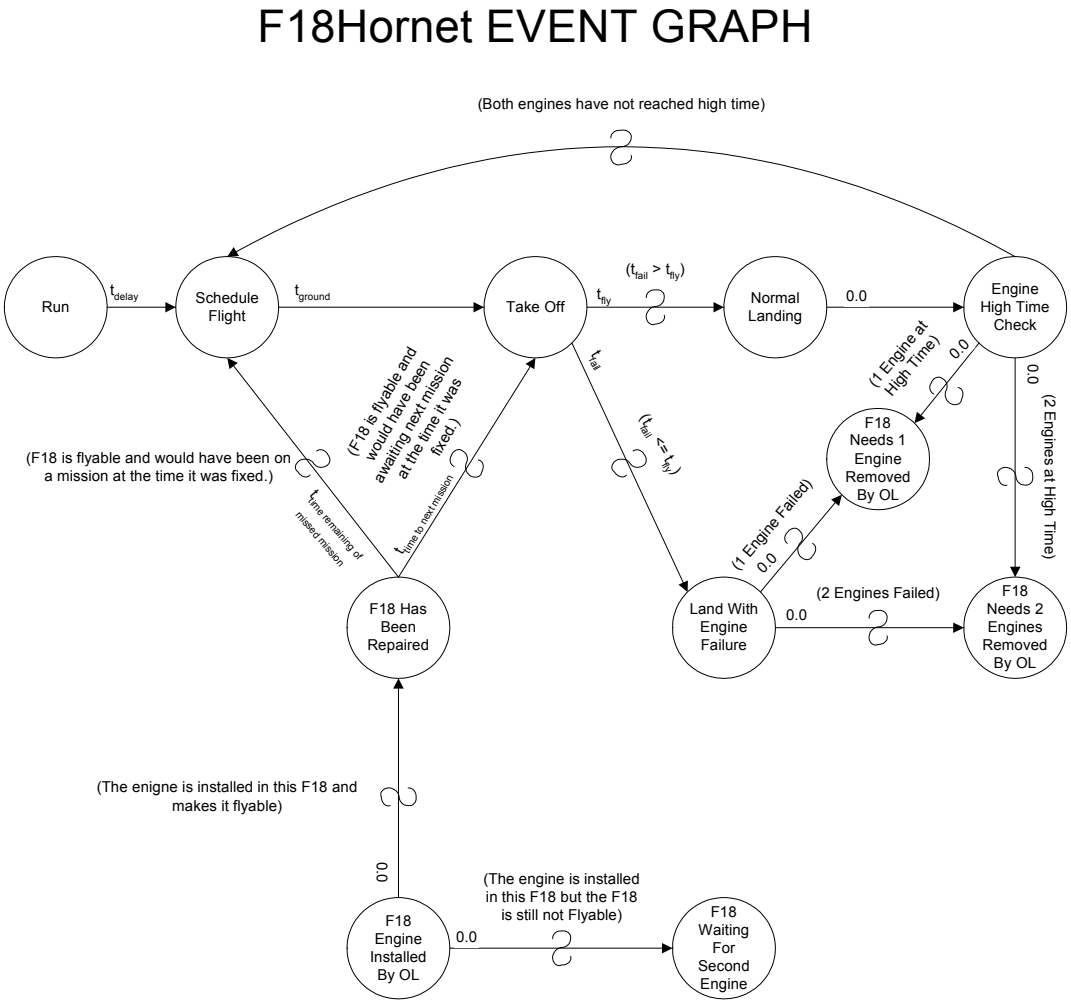


Figure 7 F18Hornet Event Graph

6. FlightSchedule

Appendix I contains the code for the *FlightSchedule* class. *FlightSchedule* is a class whose sole purpose is to provide flight schedule arrays to *F18Hornets*. It produces schedules in three different ways: custom, set, and random. The custom method lists the exact array to use. The set method develops the array using a specified block of days in a week, mission start time for each day (all flights for a single F/A-18 begin at the same time each day), and flight time. The random method creates the array based only on a set mission flight time and the number of days to fly. It randomly selects the block of days and start time to use. This information is passed to the appropriate set method for array development.

7. OLevel

Appendix J contains the code for *OLevel* class. The *OLevel* class is the computer code representation of a physical O-Level. Instantiating an *OLevel* object requires the following input:

- **uic –** The unit identification code of the O-Level.
- **cannibalize –** Indicates if the O-Level cannibalizes aircraft engines. If this variable is set to cannibalize, then a mission capable F/A-18 is produced whenever there are two non-mission capable F/A-18's with one working engine each. This leaves one F/A-18 mission capable and the other requiring two replacement engines.
- **EngineTransferTime –** The time to transfer a removed engine from an O to I-Level or vice versa.

OLevel tracks the following primary states during a simulation run:

- **needs1Engine –** A list of aircraft requiring one replacement engine. This is a first-in first-out (FIFO) queue. The aircraft that has been waiting the longest for an engine gets it first.
- **needs2Engines –** A list of aircraft requiring two replacement engines. This is a FIFO queue. When a replacement engine arrives from the I-Level and no aircraft needs just one engine, the O-Level installs the engine in the first aircraft of the needs2Engine queue. That aircraft then moves to the needs1Engine queue.

Figure 8 details the *OLevel* event graph. The *OLevel* listens to each of its supported *F18Hornets* for the F18Needs1EngineRemovedByOL and

F18Needs2EnginesRemovedByOL events. Each supported *F18Hornet* listens to their supporting *OLevel* for the F18EngineInstalledByOL event.

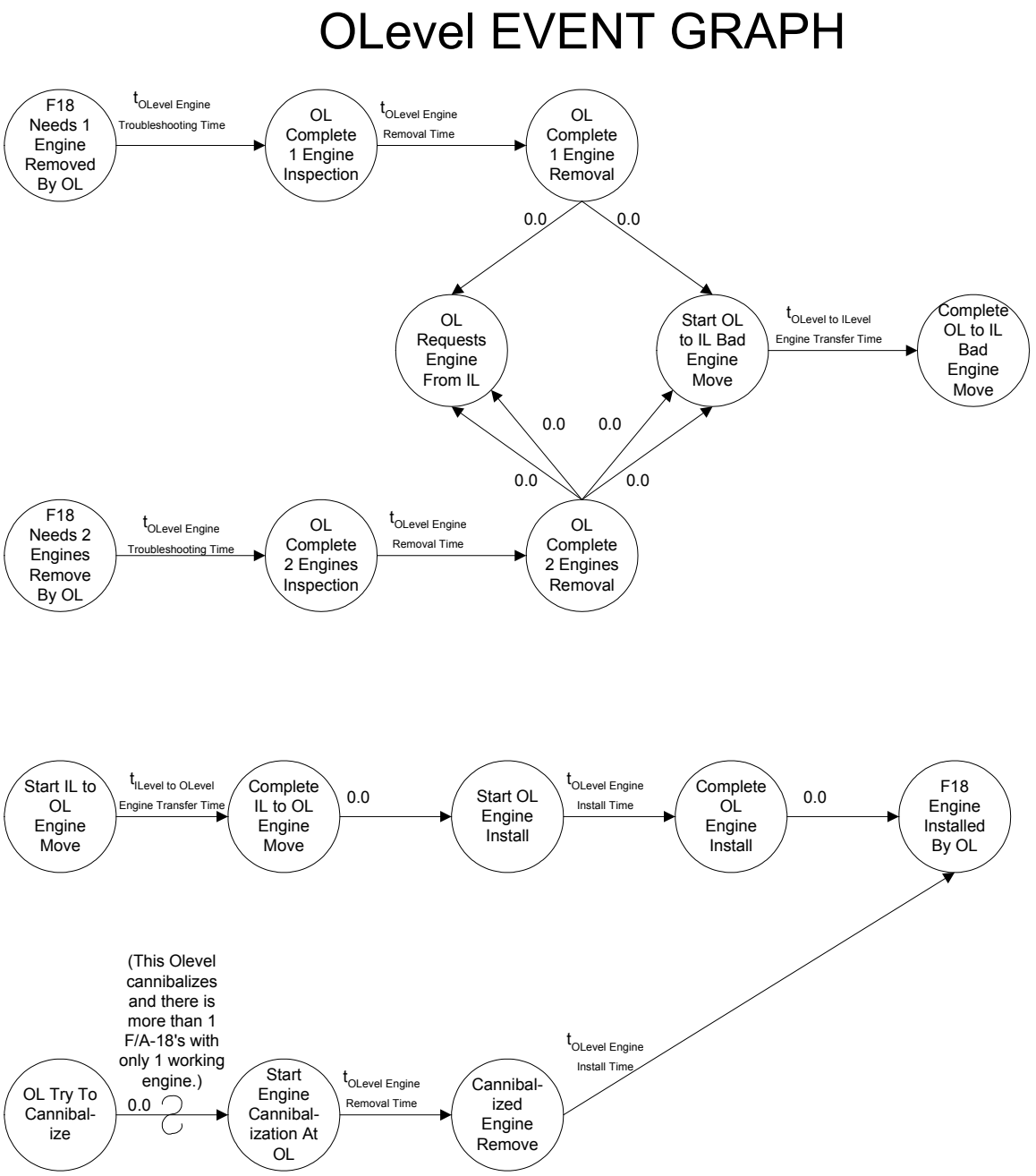


Figure 8 OLevel Event Graph

8. ILevel

Appendix K contains the code for the *ILevel* class. The *ILevel* class is the computer code representation of a physical I-Level. Instantiating an *ILevel* object requires the following input:

- **uic –** The unit identification code of the O-Level.
- **iLtoDLTransferTime -** The time to transfer a module from an I to D-Level or vice versa.
- **engineAllowance –** The number of engines the I-Level is authorized to maintain in its engine rotable pool.
- **moduleAllowance –** An array that contains the number of modules the I-Level is authorized to maintain in its module rotable pool by module type.

ILevel tracks the following primary states during a simulation run:

- **engineNIS –** The number of times a supported O-Level requested an engine and none was available in the engine rotable pool.
- **moduleNIS –** An array per module type of the number of times the I-Level checked to see if it had enough modules to build an engine and did not have at least one of each module type. This check is made every time a bad engine is received from an O-Level or good module is received from the D-Level.
- **needsEngine –** A list of O-Levels that require engines. This is a first-in first-out (FIFO) queue. The O-Level that has been waiting the longest for an engine gets it first. Every time an O-Level requests an engine, it is placed onto the needsEngine list. Thus, the same O-Level can be on the list multiple times.
- **goodEnginePool –** The engine rotable pool (good engines).
- **badEnginePool –** A pool of engines that had at least one of their modules removed for repair or maintenance and not replaced yet.
- **goodModuleList –** A array of modules by module type in the rotable pool and installed in incomplete engines. This list represents all modules that can be used for engine repair.

Figure 9 details the *ILevel* event graph. Each *ILevel* listens to their supported *OLeve*l's for CompleteOLtoILBadEngineMove and OLRequestsEngineFromIL. All supported *OLevel's* listen to their supporting *ILevel* for StartILtoOLEngineMove and OLTryToCannibalize.

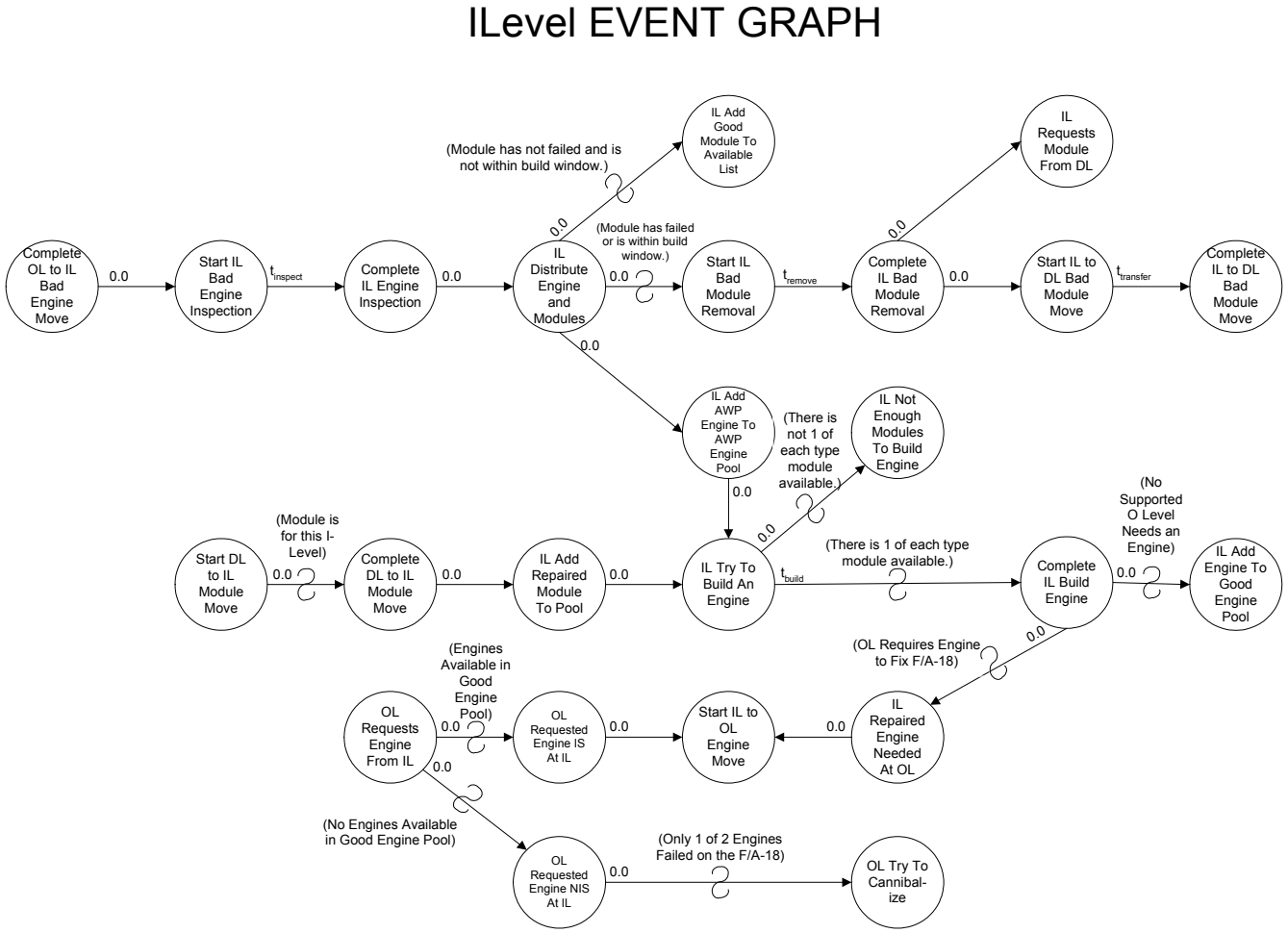


Figure 9 ILevel Event Graph

ILevel uses high time matching logic to construct engines. When the I-Level receives a bad engine, it removes all failed or high time (including those within build window time) modules from the engine. The I-Level then checks to see if a complete engine can be assembled from the partial engines and modules in the module rotable pools. If it is possible, the I-Level examines each partial engine and determines the maximum high time engine that could be produced using it and modules from other partial engines and the module rotable pool. The I-Level does this for every partial engine and then builds the engine with the greatest high time. Modules used to build the new engine are those that have the closest high times equal to or greater than the final high time of the engine when built.

Figure 10 illustrates this principal. In this example, Bad Engine #1 is received and the failed HPT removed from it. Since there are no other failed engines at this time

and no HPT's in the rotable pool, no workable engine can be produced. Bad Engine #2 is later received with a failed LPT. For the purpose of this example, it is assumed that the time to high time for the afterburner is less than the build window. The afterburner will thus be removed. Now, between Bad Engine #1, #2, and the rotable pool, there are enough modules to build an engine. Bad Engine #1 is examined first. Its constructed high time using Bad Engine #2's HPT is calculated. Bad Engine #2 is then examined. The LPT and afterburner Bad Engine #2 needs can come from either Bad Engine #1 or the rotable pool. It is determined that the best high time (assumed for the example) could be obtained by using Bad Engine #1's LPT and the afterburner from the rotable pool. The best high times for both engines if constructed are compared. It is determined that using Bad Engine #2 will result in the maximum high time engine to produce. Thus modules are pulled as stated and put into Bad Engine #2 to repair it.

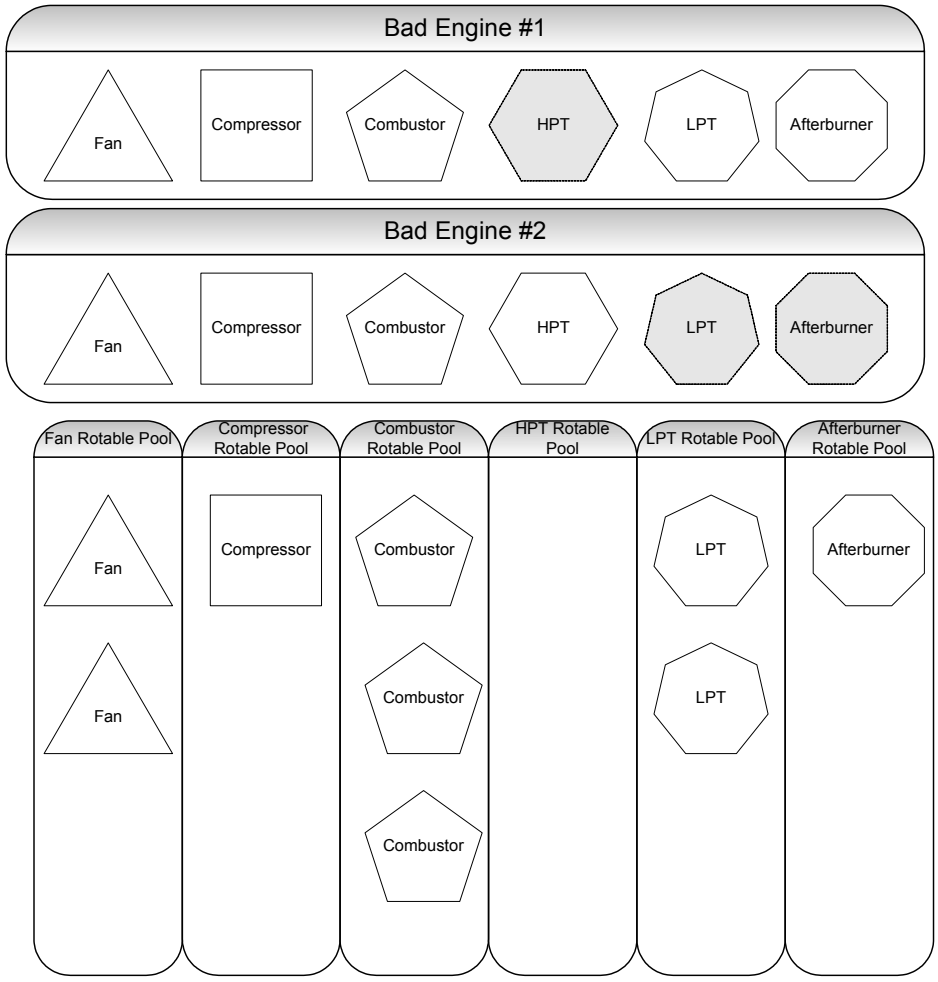


Figure 10 I-Level High Time Matching Logic Example

9. DLevel

Appendix L contains the code for the *DLevel* class. The *DLevel* class is the computer code representation of a physical D-Level. Instantiating a *DLevel* object requires the following input:

- **uic –** The unit identification code of the D-Level.
- **moduleAllowance –** An array that contains the number of modules the D-Level is authorized to maintain in its module rotable pool by module type. Although this simulation does not use a D-Level with a module rotable pool, this functionality is available for possible future research.
- **repairRV** The array of random variates from which module times between repair are drawn.

DLevel tracks the following primary states during a simulation run:

- **moduleNIS –** An array by module type of the number of times a module was requested from the D- Level and no modules were available to fill the demand.
- **needsModule –** An array by module type representing queues of those I-Level's that require a specific module type. These are FIFO queues. Every time an I-Level requests a module and none are available, it is added to the appropriate queue.
- **goodModulePool –** An array of rotable pools by module type. Although this simulation does use a D-Level with a module rotable pool, this functionality is available for possible future research.

Figure 11 details the *DLevel* event graph. Each *DLevel* listens to their supported *ILevel*'s for ILRequestsModuleFromDL and CompleteILtoDLBadModuleMove. All supported *ILevel's* listen to their supporting *DLevel* for StartDLtoILEngineMove.

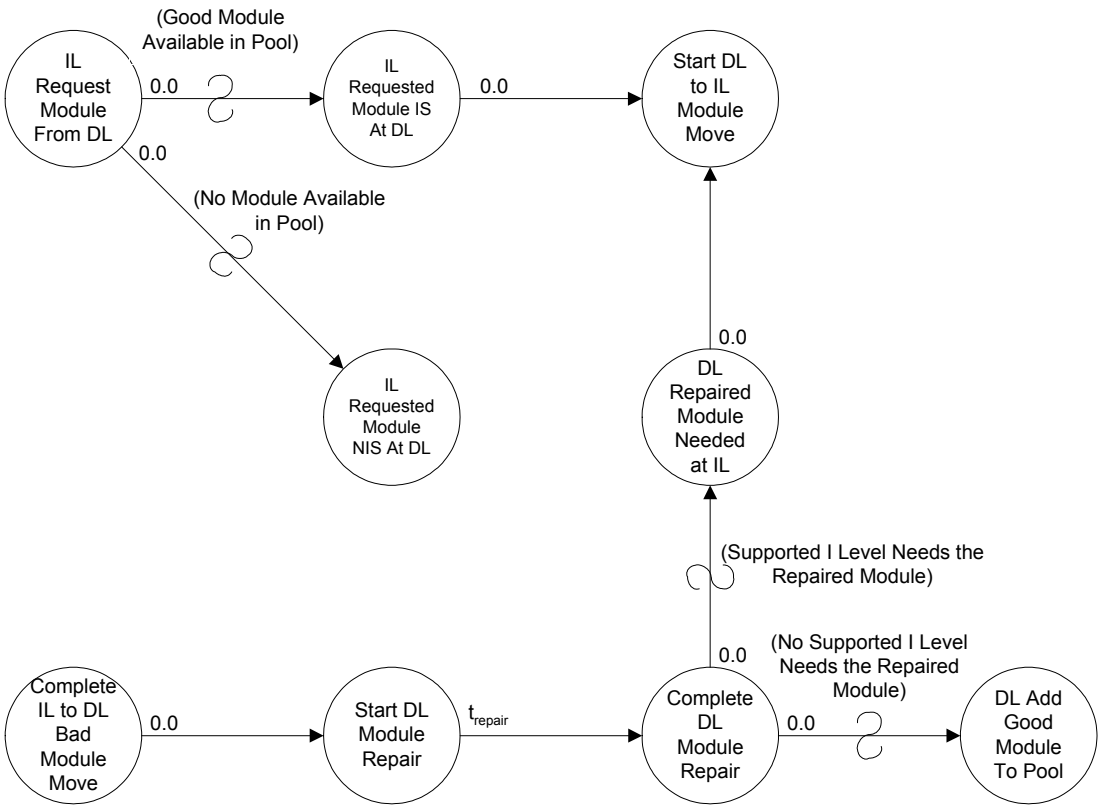


Figure 11 DLevel Event Graph

10. F18SimulationManager

Appendix M contains the code for the *F18SimulationManager* class. *F18SimulationManager* performs numerous functions within the simulation. It instantiates and tracks all *Module, Engine, F18Hornet, OLevel, ILevel,* and *DLevel* objects in the simulation based on input from *F18SimulationSetup*. It also assists in the reset of the simulation between runs, maintains annual statistics, and generates all reports within the simulation. Figure 12 details the *F18SimulationManager* event graph.

F18SimulationManager Event Graph

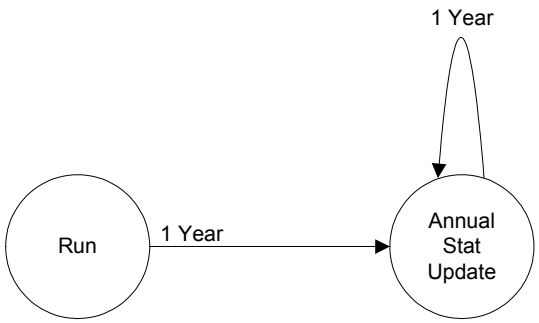


Figure 12 F18SimulationManager Event Graph

11. F18SimulationRandomness

Appendix N contains the code for the *F18SimulationRandomness* class. *F18SimulationRandomness* is the sole class of the simulation from which random variates are created. It incorporates a single random number stream for all random variables, thus ensuring independence of all generated variates.

The *F18SimulationRandomness* class uses sampling with replacement for module times between failures and depot repair turn around times. It reads in the data from text files to arrays that are usable by the random variates of Simkit. For depot repair turn around time, 24 files are used – four for each module type (AWP, AWM, In Work, and Other times). A minimum of six files are needed for module times between failures – one for each module type.

12. F18SimulationSetup

Appendix O contains the code for the *F18SimulationSetup* class. *F18SimulationSetup* is the Java class that controls the entire simulation and as such has the only main method of all twelve classes. *F18SimulationManager* sets the total infrastructure simulated and sets the parameters for the simulation run based upon the direction from *F18SimulationSetup*.

E. EXPERIMENTAL DESIGN

The goal of the thesis is to design a simulation that provides estimated operational availability and the probability to spare the repair process. The experimental design is set to exercise the simulation and demonstrate the effects that changes in various input parameters have. Each simulation run provides results from 100 repetitions of a 25-year simulation.

Chapter IV (Results and Analysis) uses Baseline1 to compare all other results. Baseline1 consists of the following input parameters:

- Table 7, Table 8, and Table 9 of Appendix A are the inputs and results NAVICP-P used for its 2002 budget input (Williamson, 11 August 2003). Appendix A provides the number of F/A-18's per activity, the number of O thru D-Levels, the engine and module sparing for each I-Level, the number of flight hours flown per aircraft per month, and the I to D-Level shipping times. Fiscal Year 2005 figures are used.
- At the O-Level, aircraft have a 2-week stagger between service entries. The maximum number of aircraft in an O-Level for Baseline1 is 25. Thus, all aircraft enter service in the first year of the simulation.
- The O-Levels cannibalize engines in Baseline1.
- Flight schedules are chosen randomly but follow a 3-day block/2.75 hours per mission pattern.

Six experiments are conducted in this thesis to represent possible management decisions that could be made by or affect NAVICP-P's supply support for the F414-GE-400. Experiments 1, 2, and 6 are policy decisions. Experiments 3, 4, and 5 are performance and/or contractor related effects that could be obtained through the expenditure of additional funds.

Experiment 1 tests the effects of staggering F/A-18 service entry. Separate simulation runs simulate zero, one, three, four, and five weeks stagger of aircraft entry (the two week result is the baseline).

Experiment 2 tests the effect of aircraft engine cannibalization at the O-Level. This experiment requires two simulation runs. The first allows cannibalization (Baseline1) and the second does not.

Experiment 3 tests the effect of improved module reliability. Separate simulation runs simulate improved reliability by adding a set number of hours onto each observation

in the module time between failure data files. The times added are 250 hours, 500 hours, 750 hours, 1000 hours, and 9999999.9 hours (to represent a non-failing module). The non-failing module simulation run shows the effects of high time removals on operational availability.

Experiment 4 tests the effect of using a mean RTAT in the simulation by changing the repair time files to reflect only one mean repair time value per module. Experiment 4's design also shows the benefit of improved turnaround time. The mean RTATs used are 25 days, 50 days, 65 days (NAVICP-P used this in Baseline1), 100 days, 150 day, and 200 days.

Experiment 5 tests the contributions of the four components of D-Level RTAT by zeroing all times in the associate repair time files when required. The first simulation run negates (sets all times in the file to zero) the effect of Other time on Baseline1. The second negates the effect of Other and AWP times. The third negates all but In Work times.

Experiment 6, the final experiment, tests the effects of build window times. Separate simulation runs use 50 hour, 250 hour, 500 hour, 750 hour, and 1000 hour build windows respectively.

The above experiments test changes to Baseline1 a single input parameter at a time. Experiments can be run that test the effect of varying multiple inputs per simulation run. It is again noted that the intent of this thesis is not to produce sparing policy, but instead to test the impact on a given sparing by varying other input parameters. Thus, no experiment was conducted to test for an "optimal" sparing. Chapter V discusses these issues as areas for further research.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. RESULTS AND ANALYSIS

A. INTRODUCTION

This chapter presents the results of Experiments 1 thru 6, provides graphic representation of the simulation output, and highlights key points of note. Appendices Q thru V provide tabular simulation output of Mean A_0, A_0 Minimum, A_0 Maximum, A_0 Variance, A_0 Standard Deviation, and P(Spare). Mean A_0 is the average of the A_0's for the 100 runs per year. A_0 Minimum is the least value of A_0 from the 100 runs per year. A_0 Maximum is the greatest value. Since the A_0's are always less that or equal to one and the tables list all values to the nearest ten thousandth (fourth significant digit), A_0 Variance usually appears as 0.0000. P(Spare) is the probability to effectively spare the repair process.

B. EXPERIMENT RESULTS

Sections 1 thru 6 below detail the effects that changes to parameters had on operational availability. P(Spare), the probability of no bare firewalls on an aircraft during the time frame under consideration, in most experiments was zero. Simulation runs with extreme changes to input parameters did have non-zero probability results, but only for the first two years or less. Table 5 lists the simulation runs by year for all six experiments that had a P(Spare) greater than zero.

Simulation Parameter Change to Baseline1	Year	P(Spare)
4 Week Staggered Service Entry	1	0.04
5 Week Staggered Service Entry	1	0.20
Time Between Failures + 250 hours	1	0.92
Time Between Failures + 500 hours	1	1.00
Time Between Failures + 500 hours	2	0.03
Time Between Failures + 750 hours	1	1.00
Time Between Failures + 750 hours	2	1.00
Time Between Failures + 1000 hours	1	1.00
Time Between Failures + 1000 hours	2	1.00
Time Between Failures + INFINITE hours	1	1.00
Time Between Failures + INFINITE hours	2	1.00
25 Day Mean Depot RTAT	1	0.22
65 Day Mean Depot RTAT	1	0.01

Table 5 P(Spare) Greater than Zero Simulation Runs

1. Effects of Staggering F/A-18 Service Entry

Experiment 1 studies the effects of staggering aircraft entry at the O-Level on operational availability. Staggering refers to a delay between the start of aircraft services. For example, a two-week stagger means that the first aircraft starts flying at time zero, the second two weeks later, the third two weeks after the second, and so on. The staggers are in one-week increments starting with no stagger. The first simulation run, zero week stagger, equates to all aircraft at each O-Level starting their flight schedule on the same week. Baseline1's two-week stagger allows for all aircraft to enter service in the first year of the simulation run since the maximum number of F/A-18's at any O-Level is 25 (two-week stagger X 25 F/A-18 maximum at the O-Level = 50 weeks for all to enter service).

Experiment 1 illustrates the effect that aircraft service entry times have on operational availability. The use of staggering and different flight schedules per aircraft is important due to the potential high time "bow wave" (surge) caused without them. If all aircraft started service and flew missions at the exact same time, there is a possibility that all or a majority of the aircraft would reach their first high time at the exactly same time. In most realistic sparings (i.e. those that are economically feasible), this would produce a "bow wave" of engine demand that could not be met.

Figure 13 displays the results of Experiment 1. Appendix Q, Table 15 to Table 20 provides the data used to generate Figure 13. It is evident from the graph that as the stagger between aircraft service entry increases, operational availability initially is higher. A higher stagger time equates to less aircraft initially flying, failing, and drawing components from rotable pools. Thus, the rotable pool and repair cycle can meet the initial demand placed upon it by a smaller aircraft population. Ultimately though, regardless of the stagger time studied, steady state operational availability approached 0.62. The simulation output indicates that the stagger of aircraft service entry has only a short-term effect on operational availability.

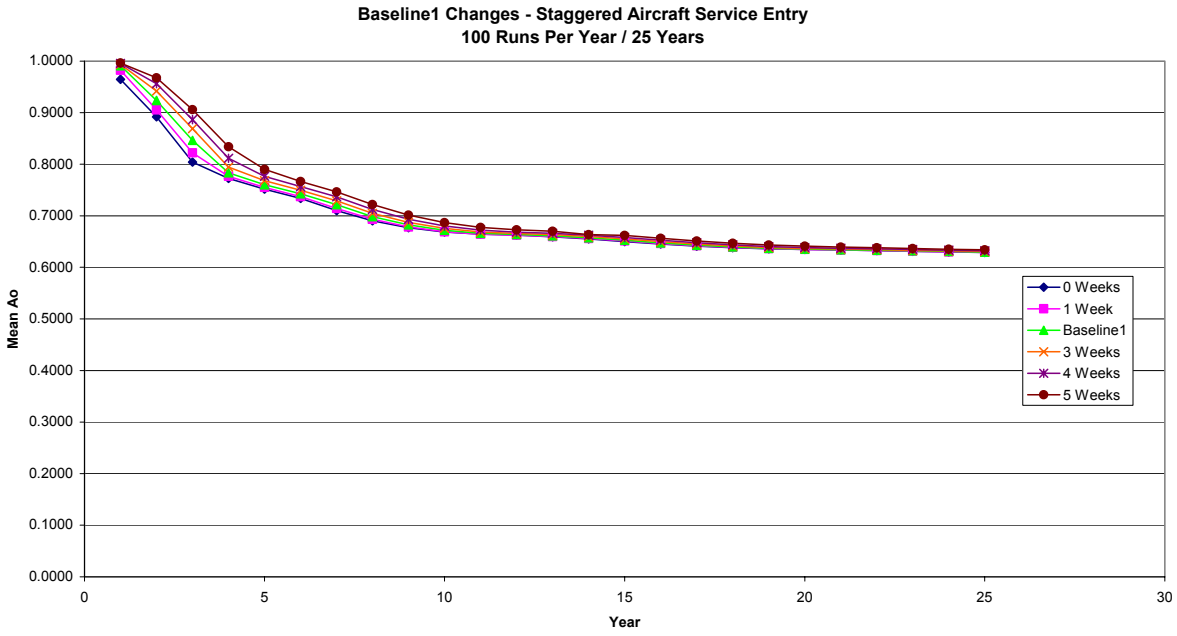


Figure 13 Baseline1 Changes - Effects of Staggering F/A-18 Service Entry

2. Effects of Engine Cannibalization at the O-Level

Experiment 2 studies the effect of the cannibalization decision at the O-Level. Cannibalization refers to using good engines off two separate non-mission capable F/A-18's to produce one mission capable aircraft. The United States Navy currently utilizes cannibalization of F414-GE-400's at the O-Level. This experiment evaluates the effectiveness on the Baseline1 scenario and the impact on operational availability if the policy was changed.

Figure 14 displays the results of the two simulation runs. The first, Baseline1, allows cannibalization at the O-Level. The second does not. The simulation assumes that no engine damage or decrease in reliability occurs during the cannibalization process. Figure 14 clearly indicates that a higher operational availability is achieved in this model by using cannibalization of engines at the O-Level. Appendix R, Table 21 and Table 22, provide the data used to generate it.

If the current policy of engine cannibalization at the O-Level is ever changed, a noticeable reduction in operational availability will occur. For Baseline1, the simulation indicates that a 0.12 decrease would be realized if cannibalization is not used.

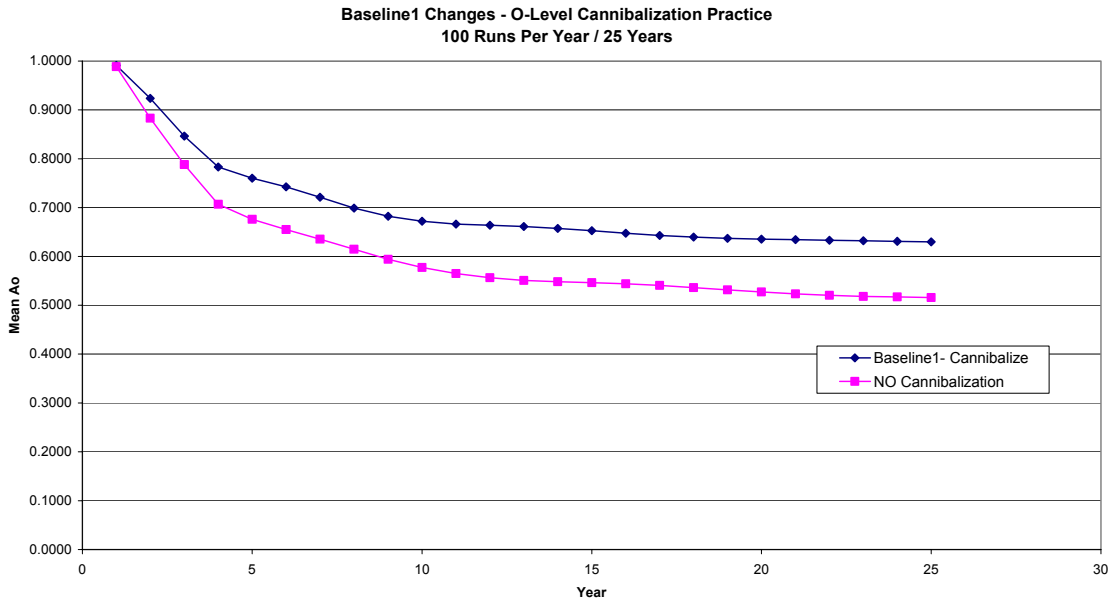


Figure 14 Baseline1 Changes - Effects of Engine Cannibalization at the O-Level

3. Effects of Improved Module Reliability

Experiment 3 studies the effect of increased module reliability and thus could assist in cost/benefit analysis of proposed contractor module improvements. This experiment is also designed to show the effects that high time removals have on operational availability. Baseline1 uses the previous three years of failure data, as discussed in Chapter II (Overview of the Data). Experiment 3 produces improved module reliability by adding a fixed value to each observation in the time between failure data files. For a 250-hour improvement, 250 hours were added to each time between failures. Experiment 3 uses 250, 500, 750, 1000, and infinite hour improvements. The simulation run approximates infinite hour improvements by adding over 1,000 years to all times between failures (only 25 years are studied in this thesis).

Figure 15 details the results of Experiment 1. Appendix S, Table 23 to Table 28 provides the data used to generate it. As reliability of the modules improves, the simulation indicates that operational availability increases. Figure 16 illustrates this trend. As reliability approaches infinity, the simulation indicates that operational availability approaches 0.83 at steady state.

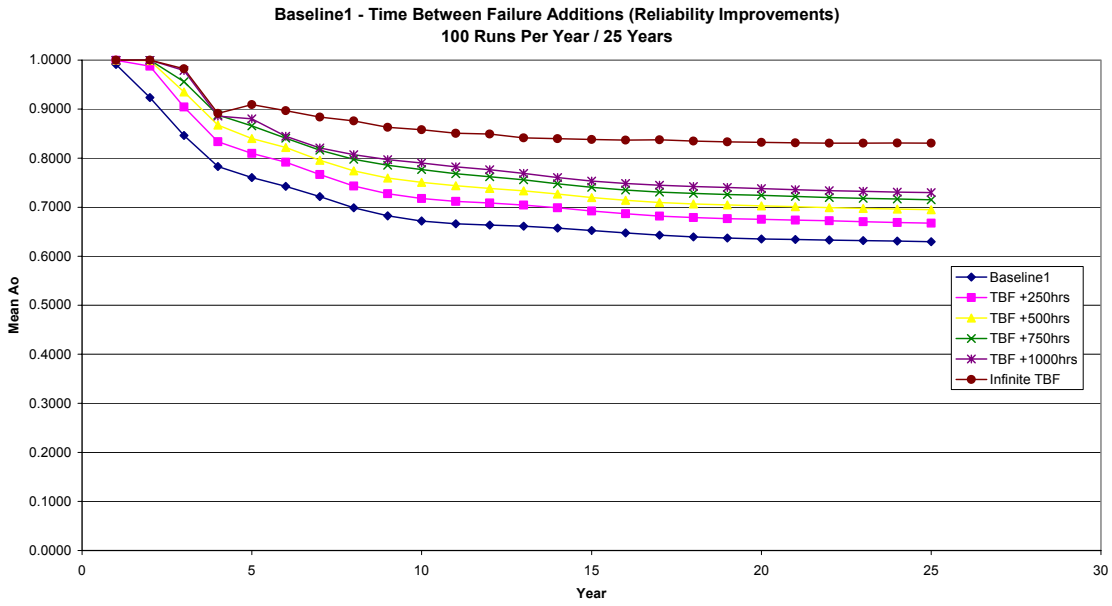


Figure 15 Baseline1 Changes – Effects of Improving Module Reliability

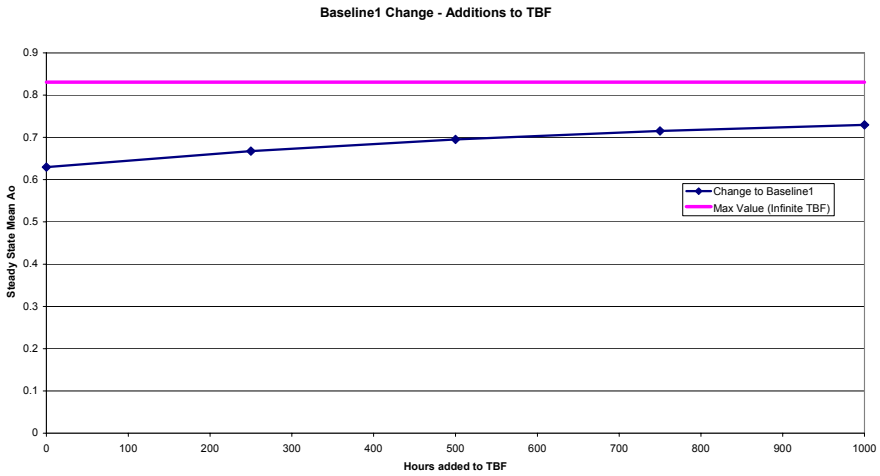


Figure 16 Baseline1 Changes – Steady State A_0 vs. Improvement to TBF

The only two reasons for engine removal in the simulation are failures and high times. The simulation run that sets times between failures to infinity precludes failure removals and only allows high time repairs. Figure 15 shows that the steady state operational availability of Baseline1 due solely to high time removals is 0.83. This also indicates that the operational availability for Baseline1 will never be above 0.83 if only improvements to modules are considered.

4. Effects of Using Mean Depot RTAT Times

Experiment 4 studies the effect of removing randomness from D-Level RTAT and using a mean value (deterministic) for each module. Experiment 4 achieves this by setting Other, AWP, and AWM D-Level repair time data file entries to zero and In Work data file entries to the mean value under consideration. Thus, the sample with replacement would always produce the mean value desired. Figure 17 displays the results of Experiment 4 (Baseline 1 is included for comparison). Appendix T, Table 29 to Table 35, provide the data used to generate it.

Experiment 4 was designed to show the effect that a set RTAT has on operational availability (i.e., if the depot could guarantee that a module would be repaired in a set amount of days). It was impossible to subtract a set amount of time from each observation (as was done with adding time in Experiment 3) due to the fact that some times were between zero and 24 hours. Subtracting 50 hours (for example) from each observation would yield some negative times. A method to subtract the set time from all observation and then make all negative observations zero was considered, but, this is invalid because the underlying distribution would be changed.

Figure 18 achieves the thesis goal of determining mean D-Level RTAT effects on operational availability. It illustrates steady state mean operational availability versus selected mean D-Level RTAT. NAVICP-P input to Baseline1 uses a 65-day mean RTAT goal. The simulation used in this thesis indicates a 0.89 steady state operational availability utilizing a 65-day RTAT (very close to the 0.90 operational availability goal that NAVICP-P wishes to obtain). To achieve a 0.90 operational availability goal, mean D-Level RTAT would need to be approximately 60 days based on the assumptions, infrastructure and sparing of Baseline1.

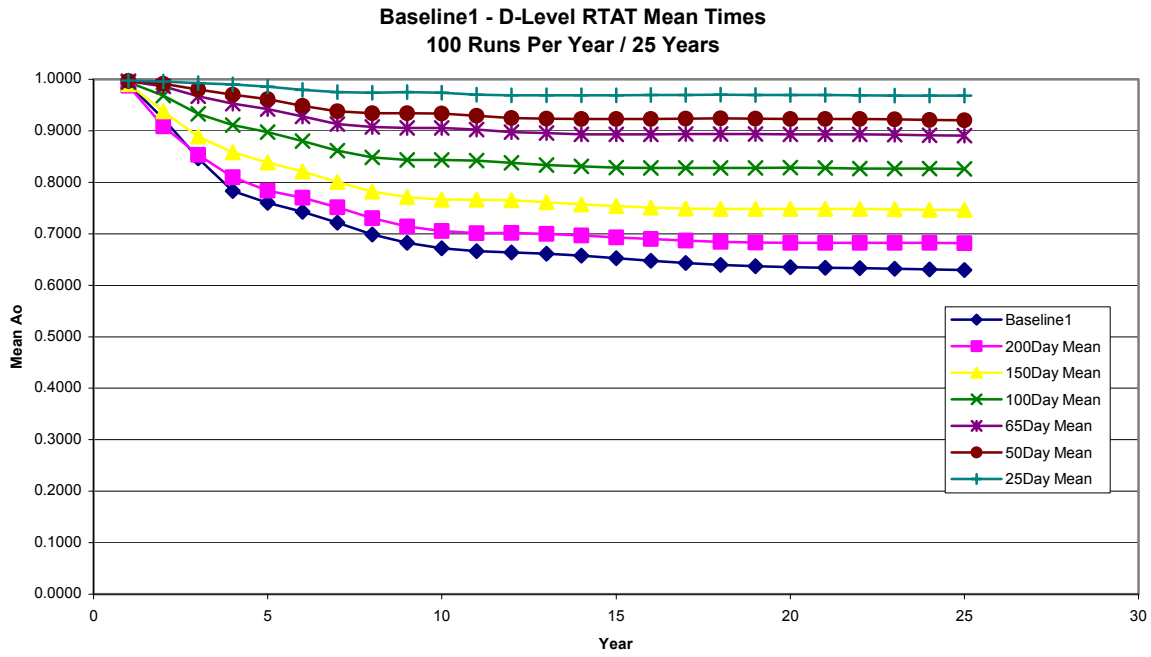


Figure 17 Baseline1 Changes – Mean D-Level RTAT Times

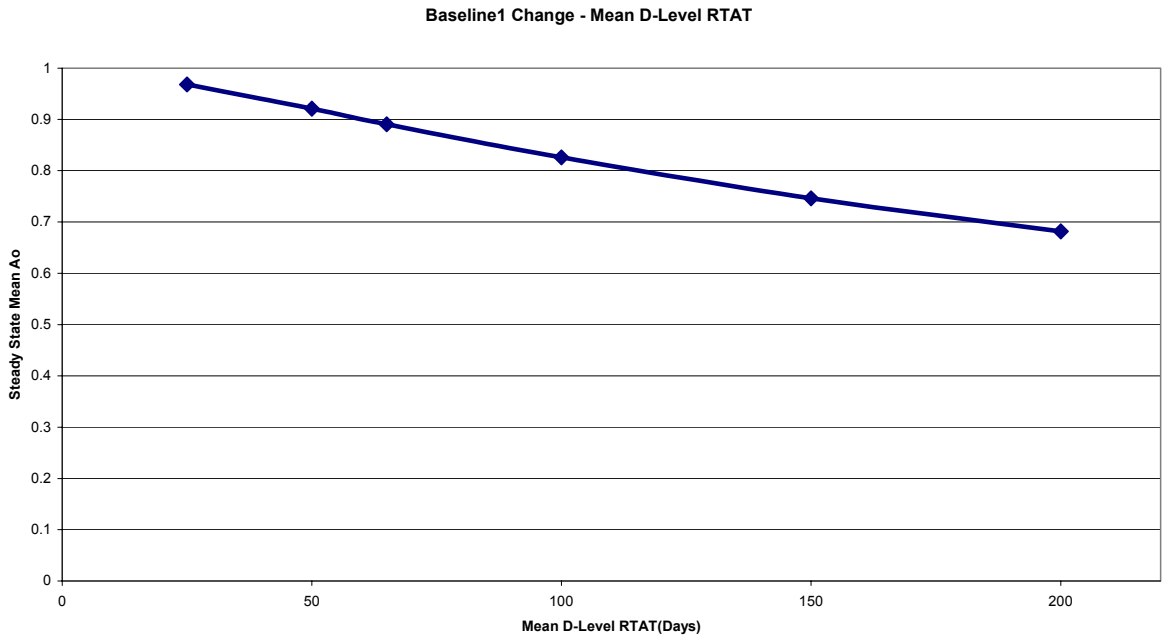


Figure 18 Baseline1 Changes – Steady State A_0 vs. Mean D-Level RTAT

5. Effects of Depot RTAT Components

Experiment 5 studies the effects that the four components of D-Level RTAT have on operational availability and addresses the question of where efforts to improve efficiencies at the D-Level should be focused. The first simulation run in Experiment 5 sets the values in the Other D-Level repair time data file entries to zero. The second run set Other and AWP times to zero. The third run has only In Work times that are non-zero. Figure 19 displays the results of Experiment 5. Appendix U, Table 36 thru Table 39, provide the data used to generate it.

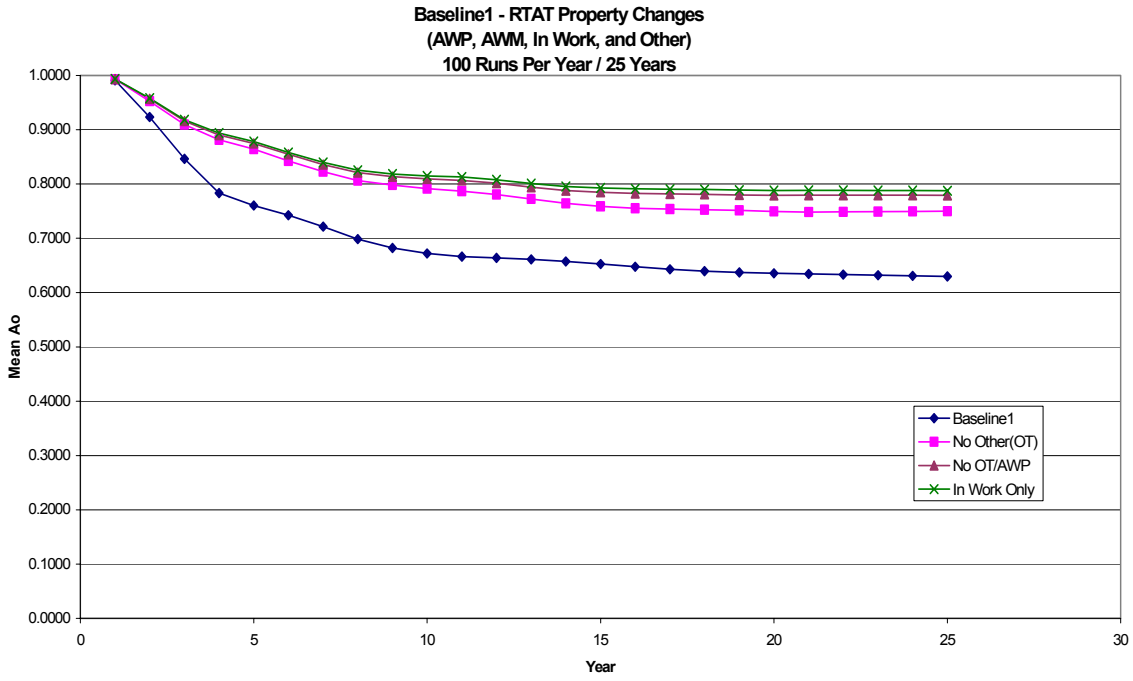


Figure 19 Baseline1 Changes – Effects of D-Level RTAT Components

Baseline1 incorporates all four components of D-Level RTAT and has a steady state operational availability of 0.63. When the effect of the Other RTAT is nullified (values set to zero), an increase of 0.12 in steady state operational availability is achieved. Even if all modules immediately started work upon arrival at the D-Level (only In Work times considered), a 0.79 operational availability is only possible for Baseline1.

Table 6 contains the mean times (rounded number of days) by module and RTAT component type for the previous three years of data. It is evident that In Work and Other times are the dominant factors of RTAT. Therefore, the prime focus of efficiency improvements in the depot repair process should concentrate on both In Work and Other time reductions.

	AWP	AWM	In Work	Other
Fan	13	8	122	169
Compressor	33	10	144	80
Combustor	17	2	117	47
HPT	10	4	115	90
LPT	25	3	125	63
Afterburner	3	1	123	143

Table 6 Mean D-Level RTAT Times in Days (Rounded)

6. Effects of Build Window Times

Experiment 6 studies the effect that changes to the length of build window would have on operational availability. Build windows of 50, 250, 500 (Baseline1), 750, and 1000 hours were considered. Figure 20 displays the results of Experiment 6. Appendix V, Table 40 thru Table 44, provide the data used to generate it.

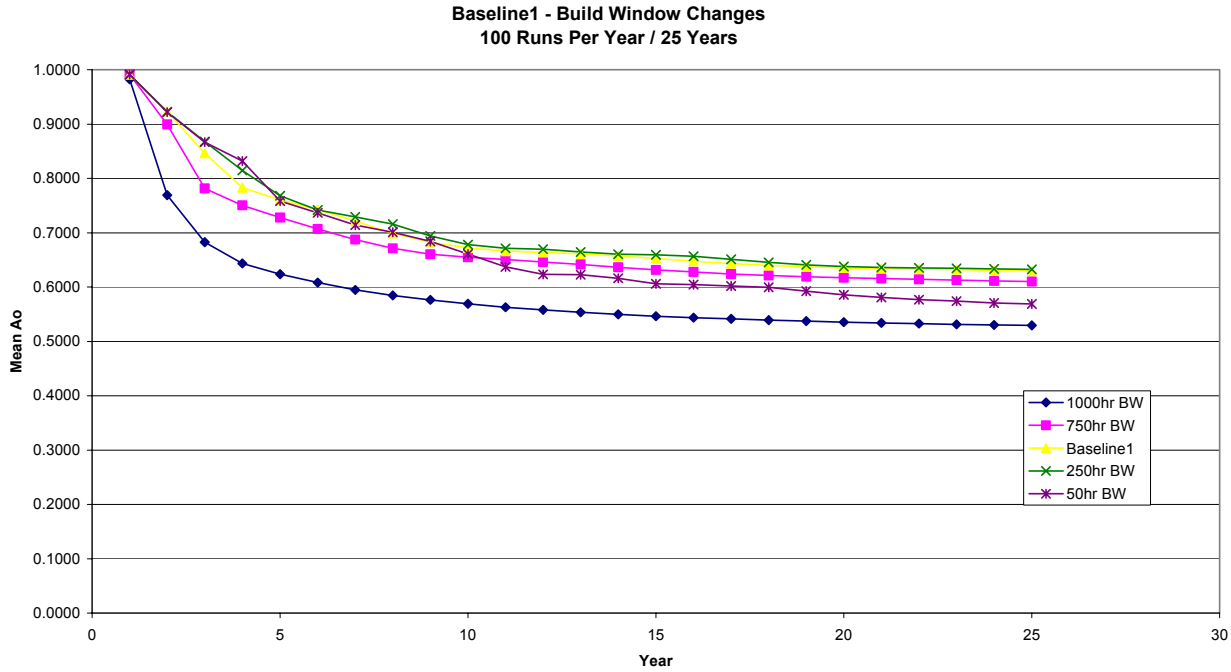


Figure 20 Baseline1 Changes – Effects of Build Window

Figure 20 displays a narrow range of operational availability for build windows between 250 and 750 hours. The 500 hour build window currently used appears to be an effective choice since it lies at the midpoint of the above range. Decreasing the build window too much initially has a slight positive impact, but over time is detrimental to operational availability. Increasing the build window too much causes excessive removals and lower operational availability from initiation of the program.

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This thesis introduces a simulation of the I3 to D Repair Process for the F414-GE-400 jet aircraft engine. The ease in changing input parameters, performing simulation runs, and obtaining pertinent results make it an excellent management and decision-making tool. It is cautioned though that only one baseline is studied in this thesis and the results contained herein may not be applicable to others (the simulation was run for specific parameters as previously discussed). As changes to sparing numbers, infrastructure, and data become available, new analysis should be performed.

A study of Baseline1 using simulation provides some excellent insight into the underlying contributing factors to operational availability. Assuming that a steady state operational availability of 0.62 or lower is not desired, Baseline1 does not have sufficient spares to support current high time maintenance, depot RTAT, peacetime flight hours, and module reliability. If additional sparing is not possible, significantly improving operational availability is only possible if improvements to all factors are considered (too myopic of an approach may be uneconomical or incapable of achieving the gains desired).

B. RECOMMENDATIONS FOR FURTHER RESEARCH

There are numerous possibilities for future research based on the foundation of this thesis. Specific ideas include:

- Research for additional baselines. As proposed baselines are produced by NAVICP-P, a detailed analysis should be completed using the techniques detailed in this thesis.
- Research the I1 to D Repair Process. In the I1 to D process, the I-Level has capabilities to repair modules. The simulation used herein is modifiable to the I1 to D Process.
- Research the effects of randomness in shipping times. A dramatic reduction to depot RTAT will cause shipping times to become a major contributing factor to operational availability.
- Research the effect of maintaining stocks of ready-for-issue modules at the depot level.

- Research the possibility of using simulation to optimize (or locally optimize) sparing. Less iteration per year per simulation run would be required to decrease the time of each run.
- Research the effect of varying multiple parameters per experiment. The experiments contained in this thesis examined the effects on operational availability caused by changes to a single parameter at a time.

Java and Simkit provide efficient and effective tools for producing complex large-scale simulations. The areas for future research they are applicable to are limited only to the imagination. The simulation used in this thesis is one applied example of a possible application.

APPENDIX A. NAVICP-P PROJECTED LEVELS

Table 7 details the number of F/A-18E/F's NAVICP-P projects to be in United States Naval Service per fiscal year. Sites 01 thru 16 are I-Level repair facilities with one supported O-Level having the number of aircraft listed.

		Projected Number of F/A-18E/F Supported						
Site ID	Site Name	FY03/SY05	FY04/SY06	FY05/SY07	FY06/SY08	FY07/SY09	FY08/SY10	FY09/SY11
Site 01	PAX River/China Lake	25	27	25	29	24	19	19
Site 02	Lemoore	28	31	30	35	49	64	64
Site 03	East Coast	20	13	22	41	61	79	79
Site 04	CV 01 (P)	26	26	26	26	26	26	26
Site 05	CV 02 (P)	26	26	26	26	26	26	26
Site 06	CV 03 (P)	12	12	12	12	12	12	12
Site 07	CV 04 (WestPac)	0	26	26	26	26	26	26
Site 08	CV 05 (L)	0	12	26	26	26	26	26
Site 09	CV 06 (L)	0	12	12	26	26	26	26
Site 10	CV 07 (L)	0	0	12	12	26	26	26
Site 11	EA-18G	0	0	0	4	12	16	33
Site 12	Atsugi	0	0	0	0	0	0	0
Site 13	Adversary	0	0	0	0	0	10	10
Site 14	NSAWC	0	0	0	0	0	12	24
Site 15	Blue Angels	0	0	0	0	0	0	10
Site 16	Reserves	0	0	0	0	0	0	10
	Total A/C Supported	**137**	**185**	**217**	**263**	**314**	**368**	**417**

Table 7 Projected F/A-18E/F's Population by Fiscal Year (After: Williamson, 11 August 2003)

Table 8 is the RIMAIR projected engine and module sparing based on the input of Table 9 and populations of Table 7.

FY05 Projected Engine and Module Sparing by Site							
	ENGINE	FAN	COMPRESSOR	COMBUSTOR	HPT	LPT	AFTERBURNER
SITE	n/a	F	C	S	H	L	A
PAX RIVER	8	7	3	4	2	1	2
NAS LEMOORE	7	8	4	5	3	1	2
NAS OCEANA	5	6	3	4	2	1	2
CV 01	8	9	5	6	4	1	2
CV 02	8	9	5	6	4	1	2
CV 03	4	5	2	4	2	0	1
CV 04	8	9	5	6	4	1	2
CV 05	8	9	5	6	4	1	2
CV 06	4	5	2	4	2	0	1
CV 07	4	5	2	4	2	0	1
EA-18G Site	0	0	0	0	0	0	0
NAF ATSUGI	0	0	0	0	0	0	0
ADVERSARY	0	0	0	0	0	0	0
NSAWC	0	0	0	0	0	0	0
BLUE ANGELS	0	0	0	0	0	0	0
RESERVES	0	0	0	0	0	0	0

Table 8 Baseline1 Module Sparing per I-Level (After: Williamson, 13 August 2003)

RIMAIR Engine and Module Parameters

Rerun of BAM04 F414 APN-6 Spares Submisson
June 24, 2002

RIMAIR Model Parameters

Engine Protection Factor	90%

Parameter	Build Up Days	Resupply Days	Support Days DEPOT
Conus	5	5	0
O-CONUS	5	15	0
CV/L-Class/MAGs	5	15	30

Weapon System Parameters

F414 Eng/Module		I-Level MTBR	Depot MTBR	I-Level RTAT	D-Level RTAT
F414 AFB		0	2639		65
F414 CMB		0	740		65
F414 DEV		0	5814		65
F414 ENG		322		34	
F414 FAN		0	429		65
F414 HPC		0	1106		65
F414 HPT		0	1786		65
F414 LPT		0	4525		65

F/A-18E/F	
A/C Utilization (Peace) Hrs/Aircraft	35
A/C Utilization (War) Hrs/Aircraft	60
Modular/Non-Modular	MOD
Engines/Aircraft	2
Percent of Maintenance Cycles to apply to engines and modules	50%

Table 9 Baseline1 NAVICP-P Initial Constrained PC ARROWS – RIMAIR Input (From: Williamson, 11 August 2003)

APPENDIX B. REMOVAL REASON CODES

CODE	REMOVAL REASON
0	
0A	COMPUTER GENERATED CODE FROM STST 1020. CLOSEOUT
1	
1A	FLAMEOUT
1B	HOT START_____
1E	BACKFIRE(S)IN FLIGHT
1G	ACCESSORY GEARBOX MALFUNCTION
1L	FOD DAMAGE
1M	SYN/ACT RING MALFUNCTION
1R	REDUCTION GEARBOX FAILURE
1S	PROP SHAFT LOOSE
1T	INTERNAL NOISE/BINDING/SHUTDOWN/START
1V	HPT BLADE SULFIDATION
1W	OIL CONTAMINATION (OTHER THAN METAL)
1Y	REMOVED FOR HIGH COUNTS
1Z	SMOKE/FUMES IN COCKPIT
2	
2A	INLET CASE CRACK
2C	INTERNAL/NOISE/BINDING/SHUTDOWN/RUB
2F	OVERSPEED
2G	OVERBOOST
2N	METAL IN OIL
2Q	CORROSION
2S	IGV CRACK/BROKEN/DISCONNECT
2U	CYLINDER FAILURE
3	
3A	CAN'T TRIM (HIGH / LOW ENGINE PRESSURE)
3B	OTHER MAINTENANCE
3D	VIBRATION
3E	FAULTY HANDLING/DROPPED
3M	INABILITY TO ACCELERATE
3P	INABILITY TO START
3Q	INTERAL FAILURE CAUSE UNKNOWN
3R	LOW/POWER/TORQUE/THRUST/EFFICIENCY/LOW FAN SPEED
3T	OIL CONSUMPTION
3U	UNSTABLE/SURGING
3W	CANNIBALIZATION
3X	CANNIBALIZATION (ADMINISTRATIVE)

Table 10 Removal Reason Codes 0-3

CODE	REMOVAL REASON
4	
4A	ASSOCIATED ENG/MOD FAILURE
4B	ACCIDENT/INCIDENT DAMAGE
4D	DIRECTED REMOVAL (RFI)
4E	TEST CELL VERIFICATION
4J	SUDDEN STOPPAGE
4L	DAMAGED IN TRANSIT
4M	DIFFUSER CRACKED
4P	ENGINE SEIZURE
4R	EXHAUSE DUCT FAILURE
4S	VARIABLE NOZZLE FAILURE
4W	RESEARCH,DEVELOPMENT,TEST & EVALUATION PROJECTS
5	
5A	FIRE FIGHTING CHEMICAL INGESTION
5B	FIRE-ENGINE
5C	COMPRESSOR FOD
5D	TURBINE FOD
5E	COMPRESSOR INTERNAL FAILURE
5F	FIRE - AIRCRAFT
5G	OVERHAUL
5H	TURBINE INTERNAL FAILURE
5I	INTERNAL FAILURE
5Q	HIGH OIL PRESSURE
5U	ENGINE DECOUPLED
5W	LOW OIL PRESSURE
6	
6A	MOD/TDC INC
6C	NOSE CASE CRACK
6E	OIL STARVATION
6F	OVERTEMP
6J	COMP VANE/BLADE/SHROUD CRACKED/BROKEN
6K	BLADE/VANE BLENDED BEYOND LIMITS
6L	INTERNAL NOISE/BINDING/SHUTDOWN RUB
6N	CVG DISCONNECT / BROKE / FAILURE
6P	COMPRESSOR CASE CRACK
6Q	COMPRESSOR STALL
6R	COMPRESSOR EROSION/CORROSION
6T	TURBINE NOZZLE FAILURE
6V	LOW CYCLE FATIGUE
6Z	COMPRESSOR VANE/BLADE/SHROUD CRACKED/BROKEN

Table 11 Removal Reason Codes 4-6

CODE	REMOVAL REASON
7	
7A	FRONT / REAR CASE CRACKED
7C	HIGH TIME COMPONENT ONLY
7D	HIGH TIME COMPONENT (HEMP / HSI)
7E	HEMP/HSI (INSP ONLY)
7G	HIGH TIME COMPONENT (COUNTS)
7I	SALT WATER INGESTION
7J	AIR LEAKAGE
7K	OIL LEAKAGE
7L	FUEL LEAKAGE
7P	BASIC ENGINE - RUB
7T	BLADE BROKEN/CRACKED
7U	BLADE EROSION
8	
8A	EXCESSIVE OIL FROM BREATHER
8B	TEMP OUT OF LIMITS
8C	TURBINE DISK FAILURE
8D	COOLING AIR HOLES CLOGGED
8E	NOZZLE SEGMENTS CRACKED / WORN
8F	JOAP LAB RECOMMENDATION
8K	FLAME HOLDER CRACKED
8N	BLEED VALUE BROKEN/INOPERATIVE
8P	BEARING FAILURE
8Q	MOUNT CRACK/BROKEN
8T	LINER CRACK/FAILURE
8V	NOZZLE CYCLINDER FAILURE/CRACK
8X	NOZZLE FLAP/SEAL/CRACK/WORN
8Y	AFTERBURNER FAILED TO LIGHT
9	
9A	TS/PENALTY/TEST CELL RUN
9B	WORN AIRSEAL
9C	FAN FOD
9D	TURBINE INTERNAL FAILURE
9E	COMB CASE CRACK/WORN
9F	WET START
9G	BLADE BROKEN/CRACKED
9H	COMB LINER CRACK/WORN
9J	HIGH SUMP PRESSURE
9K	BLADE BLENDED BEYOND LIMITS
9M	COOLING AIR HOLES CLOGGED
9N	INLET CASE VG'S CRACKED / BROKEN
9P	FAN CASE CRACKED
9Q	BLADE EROSION
9R	NOZZLE FAILURE
9S	MATERIAL MISSING
9T	INTERNAL NOISE/BINDING
9U	DISK FAILURE
9V	LPT SULFIDATION
9W	SHROUD CRACKED/WORN/MISSING
9X	SHROUD CRACKED/WORN/MISSING
9Y	SPRAYBAR CRACK/FAIL
9Z	AFTERBURNER FUEL LEAK

Table 12 Removal Reason Codes 7-9

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. F414-GE-400 MODULE HIGH TIMES

Module	High Time	Base Units	Conversion Factor	High Time (EOT)
FAN	1100	EOT	1.0	1,100
	2000	EOT	1.0	2,000
	2200	EOT	1.0	2,200
	4000	EOT	1.0	4,000
	6420	ELCF	1.2	5,350
	8610	ELCF	1.2	7,175
	8810	ELCF	1.2	7,342
	9090	ELCF	1.2	7,575
	12840	ELCF	1.2	10,700
COMPRESSOR	2000	EOT	1.0	2,000
	3560	ELCF	1.2	2,967
	3690	ELCF	1.2	3,075
	4560	ELCF	1.2	3,800
	4000	EOT	1.0	4,000
	4980	ELCF	1.2	4,150
	5660	ELCF	1.2	4,717
	9740	ELCF	1.2	8,117
COMBUSTOR	2000	EOT	1.0	2,000
HPT	1444	ELCF	1.2	1,203
	1850	ELCF	1.2	1,542
	2640	ELCF	1.2	2,200
	2900	ELCF	1.2	2,417
	2980	ELCF	1.2	2,483
LPT	2000	EOT	1.0	2,000
	4000	EOT	1.0	4,000
	5090	ELCF	1.2	4,242
	19950	ELCF	1.2	16,625
	22830	ELCF	1.2	19,025
AFTERBURNER	2000	EOT	1.0	2,000

Table 13 F414-GE-400 Module High Times (After: Williamson, 30 April 2003)

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. MODULETYPE CLASS JAVA CODE

```
/**
 * LCDR Eric J. Schoch
 * F-18 Thesis Work
 * Last Updated 21JUL03
 * <P>
 * Comments: A class that sets the types on modules that can be used in the
 * simulation.
 **/

package jet;
import java.util.*;

public class ModuleType {

  //CLASS VARIABLES
  private static Map moduleTypes = new HashMap(6);

  public static final ModuleType FAN = new ModuleType("Fan");
  public static final ModuleType COMPRESSOR = new ModuleType("High Pressure Compressor");
  public static final ModuleType COMBUSTOR = new ModuleType("Combustor");
  public static final ModuleType HPT = new ModuleType("High Pressure Turbine");
  public static final ModuleType LPT = new ModuleType("Low Pressure Turbine");
  public static final ModuleType AFTERBURNER = new ModuleType("Afterburner");

  //INSTANCE VARIABLES
  private String name;

  //CLASS METHODS
  public static ModuleType getModuleType(String name) {
    return (ModuleType) moduleTypes.get(name);
  }

  //CONSTRUCTOR METHODS
  protected ModuleType(String name) {
    this.name = name;
    moduleTypes.put(name,this);
  }

  //INSTANCE METHODS
  public String toString() { return name; }

}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. MODULE CLASS JAVA CODE

```
/**
 * LCDR Eric J. Schoch
 * F-18 Thesis Work
 * Last Update: 12 AUG 03
 * <P>
 * Comments: A class to simulate a single module of an aircraft engine.
 **/

package jet;
import simkit.random.*;

public class Module {

  //INSTANCE VARIABLES
  //INSTANCE VARIABLES - Modules Description Variables
  //   type = A specific module type from the ModuleType class.
  //   serno = An integer serial number specific to this module.
  //   NOTE: type with serno describes a specific module.
  //   timeOnModule = total run time on the module
  //   timeBetweenRepairs = time since last failure or high time repair
  //   isUsable = Indicates if module has not failed or reached high time.
  //   engineIn = Indicates the engine this module is installed in.  A null
  //                  value indicates it is not installed in an engine.
  private ModuleType type;
  private int serno;
  private double timeOnModule;
  private double timeBetweenRepair;
  private boolean isUsable;
  private Engine engineIn;

  //INSTANCE VARAIABLES - High Time Variables
  //   timeToHighTimes = How much time before a high time repair is needed.
  //   closestHighTime = The least of all timeToHighTimes.
  //   reachedHighTime = Indicates if this module needs high time repair.
  private double[] timeToHighTimes;
  private double closestHighTime;
  private boolean reachedHighTime;

  //INSTANCE VARIABLES - Failure Variables
  //   rv = An array of RandomVariates from which failure times are pulled.
  //   rvPointer = Points to what RandomVariate in rv to use.
  //   timeToFailure = How much time before a failure repair is needed.
  //   reachedFailure = Indicates if this module needs failure repair.
  private RandomVariate[] rv;
  private int rvPointer;
  private double timeToFailure;
  private boolean reachedFailure;

  //CONSTRUCTOR METHODS
  public Module (ModuleType typ, int srn, RandomVariate[] r) {
```

```
    //Module Description Variables in the Constructor
    this.type = typ;
    this.serno = srn;
    timeOnModule = 0.0;
    timeBetweenRepair = 0.0;
    isUsable = true;
    engineIn = null;
    //High Time Variables in the Constructor
    timeToHighTimes = EngineBlueprint.getModuleHighTimes(this.getModuleIndex());
    this.updateClosestHighTime();
    if (closestHighTime <= 0.0) {
      throw new IllegalArgumentException("Module Class Constructor Method - " +
        "closestHighTime on new module was less than zero.");
    }
    else {
      reachedHighTime = false;
    }
    //Failure Variables in the Constructor
    if (r.length < 1) {
      throw new IllegalArgumentException("Module Class Constructor Method - " +
        "must have at least one failure random variate.");
    }
    rv = new RandomVariate[r.length];
    rvPointer = 0;
    for (int index = 0; index < rv.length; index++) {
      this.rv[index] = r[index];
    }
    this.updateNextFailureTime();
    reachedFailure = false;

  }

  //INSTANCE METHODS
  //INSTANCE METHODS - to reset the simulation
  public void simulationReset() {
    timeOnModule = 0.0;
    timeBetweenRepair = 0.0;
    isUsable = true;
    engineIn = null;
    timeToHighTimes = EngineBlueprint.getModuleHighTimes(this.getModuleIndex());
    this.updateClosestHighTime();
    reachedHighTime = false;
    rvPointer = 0;
    this.updateNextFailureTime();
    reachedFailure = false;
  }

  //INSTANCE METHODS - Getter Methods
  //INSTANCE METHODS - Getter Methods - Module Description Variables

  public ModuleType getType(){ return type; }

  public int getModuleIndex() { return EngineBlueprint.getIndexOfModuleType(type); }

  public int getSerno() { return serno; }
```

```
public double getTimeOnModule() { return timeOnModule; }

public double getTimeBetweenRepair() { return timeBetweenRepair; }

public boolean getIsUsable() { return isUsable; }

public Engine getEngineIn() { return engineIn; }

public boolean getInstalledInEngine() {
  boolean installedInEngine = true;
  if (engineIn == null) {
    installedInEngine = false;
  }
  return installedInEngine;
}

//INSTANCE METHODS - Getter Methods - High Time Variables

public double[] getTimeToHighTimes() { return timeToHighTimes; }

public double getClosestHighTime() { return closestHighTime; }

public boolean getReachedHighTime() { return reachedHighTime; }

//     Instance Methods - Getter Methods - Failure Variables

public RandomVariate[] getRV() { return rv; }

public int getRVPointer() { return rvPointer; }

public double getTimeToFailure() { return timeToFailure; }

public boolean getReachedFailure() { return reachedFailure; }

// When the module is first constructed it calls getNextFailureTime() which
// causes the rvPointer to be 1.  Since there are no failures yet, the number
// of failures is thus rvPointer - 1.  This trend continues.  Thus, for a usable
// module, the number of failures is rvPointer - 1.  rvPointer is updated after initial
// construction only when repair is done.  Thus, if the module is in the repair
// process but not yet repaired, it has not update the rvPointer.  Since rvPointer
// is one fail ahead anyway, rvPointer would thus correctly indicate the number
// of failures for a module that is not usable.
public int getNumberOfFailures() {
  int numberOfFailures = 0;
  if (isUsable) {
    numberOfFailures = rvPointer - 1;
  }
  else {
    numberOfFailures = rvPointer;
  }
  return numberOfFailures;
}

//INSTANCE METHODS - Setter Methods

//INSTANCE METHODS - Setter Methods - Module Description Variables
```

```
public void setEngineIn(Engine eng) {
  this.engineIn = eng;
}

//INSTANCE METHODS - Other Instance Methods

public void updateClosestHighTime() {
  double storage = Double.MAX_VALUE;
  for (int index = 0; index < timeToHighTimes.length; index++) {
    if (timeToHighTimes[index] < storage) {
      storage = timeToHighTimes[index];
    }
  }
  closestHighTime = storage;
}

public void updateIsUsable() {
  if (!reachedFailure && !reachedHighTime) {
    isUsable = true;
  }
  else {
    isUsable = false;
  }
}

public void updateNextFailureTime() {
  if (rvPointer < rv.length){
    timeToFailure = rv[rvPointer].generate();
  }
  else {
    timeToFailure = rv[rv.length - 1].generate();
  }
  rvPointer++;
  if (timeToFailure <= 0.0) {
    throw new IllegalArgumentException("Module Class updateNextFailureTime Method - " +
      "timeToFailure randomly drawn was less than or equal to zero.");
  }
}

public void repairModule() {
  int modIndex = this.getModuleIndex();
  // This module is getting repaired.  Thus reset the time between repair.
  timeBetweenRepair = 0.0;
  // See if the module has failed and needs failure repair.
  if (timeToFailure <= EngineBlueprint.getFailureDependencyTime(modIndex)){
    this.updateNextFailureTime();
    reachedFailure = false;
  }
  // See if the module is at high time and needs high time repair.
  for (int index = 0; index < timeToHighTimes.length; index++) {
    if (timeToHighTimes[index] <= EngineBlueprint.getBuildWindow(modIndex)){
      timeToHighTimes[index] = EngineBlueprint.getSpecificHighTime(modIndex, index);
    }
  }
  this.updateClosestHighTime();
```

```
    if (closestHighTime <= 0.0) {
      throw new IllegalArgumentException("Module Class repairModule Method - " +
        "closestHighTime on newly repaired module was less than zero.");
    }
    else {
      reachedHighTime = false;
    }
    this.updateIsUsable();
    if (!isUsable) {
      throw new IllegalArgumentException("Module Class repairModule Method - " +
        "Module was repaired but not usable.");
    }
  }

  public void useModule(double timeUsed) {
    if (timeUsed < 0.0) {
      throw new IllegalArgumentException("Module Class useModule Method - " +
        "timeUsed less than zero.");
    }
    if (!isUsable) {
      throw new IllegalArgumentException("Module Class useModule Method - " +
        "module is unusable.");
    }
    if (engineIn==null) {
      throw new IllegalArgumentException("Module Class useModule Method - " +
        "module is not part of an engine.");
    }
    if ((closestHighTime < 0.0) || (reachedHighTime)){
      throw new IllegalArgumentException("Module Class useModule Method - " +
        "cannot use a module that has previously reached high time.");
    }
    if ((timeToFailure < 0.0) || (reachedFailure)){
      throw new IllegalArgumentException("Module Class useModule Method - " +
        "cannot use a module that has previously reached failure.");
    }
    timeOnModule += timeUsed;
    timeBetweenRepair += timeUsed;
    for (int index = 0; index < timeToHighTimes.length; index++) {
      timeToHighTimes[index] -= timeUsed;
    }
    this.updateClosestHighTime();
    timeToFailure -= timeUsed;

    if (closestHighTime <= 0.0) {
      reachedHighTime = true;
    }

    if (timeToFailure <= 0.0) {
      reachedFailure = true;
    }

    this.updateIsUsable();

  }

}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. ENGINEBLUEPRINT CLASS JAVA CODE

```
/**
 * LCDR Eric J. Schoch
 * F-18 Thesis Work
 * 07 AUG 03
 * <P>
 * Comments: This class sets the exact construction of an engine using modules
 * contained within the ModuleType Class.  Every engine in the simulation will
 * conform to the engine blueprint.  This class also sets the build windows and
 * a failure dependency factor for each module.
 **/

package jet;

public class EngineBlueprint {

  //CLASS VARIABLES
  //CLASS VARIABLES - Engine Specific
  //Times taken from D.E. Sterns thesis "Logistics Simulation Metamodel for
  //F404-GE-400 Engine Maintenance" of December 1998.  Values below are the
  //mean values.
  private static final double engineOLTroubleshootTime = 3.34;
  private static final double engineOLRemovalTime = 4.01*1.63/2.0;
  private static final double engineOLInstallTime = 4.01*1.63/2.0;
  private static final double engineILInspectionTime = 4.50;
  //Values for engineBuildupTime, inconusResupply, and oconusResupply provided
  //by NAVICP Philadelphia.
  private static final double engineBuildUpTime = 5.0 * 24.0;
  private static final double inconusResupply = 5.0 * 24.0;
  private static final double oconusResupply = 15.0 * 24.0;

  //CLASS VARIABLES - Module Specific
  //The following sets the number and specific module type of each module in
  //the engine.
  private static final ModuleType[] moduleOrder = new ModuleType[] {
    ModuleType.FAN,
    ModuleType.COMPRESSOR,
    ModuleType.COMBUSTOR,
    ModuleType.HPT,
    ModuleType.LPT,
    ModuleType.AFTERBURNER };
  private static final int numberOfModules = moduleOrder.length;
  //The following sets the build windows for each module.
  private static final double[] buildWindow = new double[] {
    500.0,   // Fan
    500.0,   // Compressor
    500.0,   // Combustor
    500.0,   // HPT
    500.0,   // LPT
    500.0 }; // Afterburner
  //Failure dependency time is a length of time that if one module fails in an
  //engine and another is within the failure dependency time of failinging, it
  //fails also.
```

```
private static final double[] failureDependencyTime = new double[] {
  20.0,   // Fan
  20.0,   // Compressor
  20.0,   // Combustor
  20.0,   // HPT
  20.0,   // LPT
  20.0 }; // Afterburner
//High times provided by NAVICP Philadelphia.
private static final double[][] moduleHighTimes = new double [][] {
  new double[] {1100.0, 2000.0, 2200.0, 4000.0, 5350.0, 7175.0, 7342.0, 7575.0, 10700.0}, //Fan
  new double[] {2000.0, 2967.0, 3075.0, 3800.0, 4000.0, 4150.0, 4717.0, 8117.0}, //Compressor
  new double[] {2000.0}, //Combustor
  new double[] {1203.0, 1542.0, 2200.0, 2417.0, 2483.0}, //HPT
  new double[] {2000.0, 4000.0, 4242.0, 16625.0, 19025.0}, //LPT
  new double[] {2000.0} }; //Afterburner
//Removal times from Stearns for AIMD afloat.  Mean times for combined removal
//and installation given.  Assumption was made to divide these in half to get
//individual removal and installation times.
private static final double[] moduleRemovalTime = new double[] {
  2.40,   // Fan
  2.71,   // Compressor
  1.43,   // Combustor
  2.27,   // HPT
  1.31,   // LPT
  1.60 }; // Afterburner
//As with removal times, install times from Stearns.
private static final double[] moduleInstallTime = new double[] {
  2.40,   // Fan
  2.71,   // Compressor
  1.43,   // Combustor
  2.27,   // HPT
  1.31,   // LPT
  1.60 }; // Afterburner

//CLASS METHODS
public static double getEngineOLTroubleshootTime() { return engineOLTroubleshootTime; }

public static double getEngineOLRemovalTime() { return engineOLRemovalTime; }

public static double getEngineOLInstallTime() { return engineOLInstallTime; }

public static double getEngineILInspectionTime() { return engineILInspectionTime; }

public static double getEngineBuildUpTime() { return engineBuildUpTime; }

public static double getInconusResupply() { return inconusResupply; }

public static double getOconusResupply() { return oconusResupply; }

public static int getNumberOfModules() { return numberOfModules; }

public static int getIndexOfModuleType(ModuleType type) {
  int storage = 999999;
  for (int index = 0; index < numberOfModules; index++) {
    if (moduleOrder[index] == type) {
      storage = index;
```

```
      }
    }
    if (storage==999999) {
      throw new IllegalArgumentException("EngineBlueprint Class getIndexOfModuleType Method - " +
        "ModuleType does not exist in this EngineBlueprint.");
    }
    else {
      return storage;
    }
  }

  public static ModuleType getModuleTypeOfIndex(int index) {
    if ((index >= numberOfModules) || (index < 0)) {
      throw new IllegalArgumentException("EngineBlueprint Class getModuleTypeOfIndex Method - " +
        "Index requested does not exist.");
    }
    else {
      return moduleOrder[index];
    }
  }

  public static double getBuildWindow(int index) {
    if ((index >= numberOfModules) || (index < 0)) {
      throw new IllegalArgumentException("EngineBlueprint Class getBuildWindow Method - " +
        "Index requested does not exist.");
    }
    else {
      return buildWindow[index];
    }
  }

  public static double getFailureDependencyTime(int index) {
    if ((index >= numberOfModules) || (index < 0)) {
      throw new IllegalArgumentException("EngineBlueprint Class getFailureDependencyTime Method - "
+
        "Index requested does not exist.");
    }
    else {
      return failureDependencyTime[index];
    }
  }

  public static double[] getModuleHighTimes(int index) {
    if ((index >= numberOfModules) || (index < 0)) {
      throw new IllegalArgumentException("EngineBlueprint Class getModuleHighTimes Method - " +
        "Index requested does not exist.");
    }
    else {
      return (double[]) moduleHighTimes[index].clone();
    }
  }

  public static double getSpecificHighTime(int moduleIndex, int highTimeIndex) {
    if ((moduleIndex >= numberOfModules) || (moduleIndex < 0)) {
      throw new IllegalArgumentException("EngineBlueprint Class getSpecificHighTime Method - " +
        "Module Index requested does not exist.");
```

```
    }
    else if ((highTimeIndex >= moduleHighTimes[moduleIndex].length) || (highTimeIndex < 0)) {
      throw new IllegalArgumentException("EngineBlueprint Class getSpecificHighTime Method - " +
        "Hightime Index requested does not exist.");
    }
    else {
      return moduleHighTimes[moduleIndex][highTimeIndex];
    }
  }

  public static double getModuleRemovalTime(int index) {
    if ((index >= numberOfModules) || (index < 0)) {
      throw new IllegalArgumentException("EngineBlueprint Class getModuleRemovalTime Method - " +
        "Index requested does not exist.");
    }
    else {
      return moduleRemovalTime[index];
    }
  }

  public static double getModuleInstallTime(int index) {
    if ((index >= numberOfModules) || (index < 0)) {
      throw new IllegalArgumentException("EngineBlueprint Class getModuleInstallTime Method - " +
        "Index requested does not exist.");
    }
    else {
      return moduleInstallTime[index];
    }
  }

}
```

APPENDIX G. ENGINE CLASS JAVA CODE

```
/**
 * LCDR Eric J. Schoch
 * F-18 Thesis Work
 * Last Update: 09 AUG 03
 * <P>
 * Comments: A class to simulate an aircraft engine which is made up of specific
 * modules (from the Module class).  Any engine created must conform to the
 * EngineBlueprint.
 **/

package jet;

public class Engine {

  //INSTANCE VARIABLES
  //INSTANCE VARIABLES - Engine Decriptive Variables
  //     serno = An integer serial number specific to this engine.
  //     numberInstalledModules = The number of modules currently installed in
  //                              this engine.
  //     modules  = The array of modules that make up this engine.
  //     numberOfModules = How many modules the EngineBluprint class says should
  //                       be in this engine.
  //     timeOnEngine = total run time on the engine
  //     timeBetweenRepair = time since last failure or high time repair
  //     isUsable = Indicates if the engine has not failed or any module on the
  //                engine has not reached high time.
  private int serno;
  private int numberInstalledModules;
  private Module[] modules;
  private int numberOfModules;
  private double timeOnEngine;
  private double timeBetweenRepair;
  private boolean isUsable;

  //INSTANCE VARIABLES - High Time Variables
  //     timeToHighTime = The least timeToHighTime of all this engines modules.
  //     reachedHighTime = Indicates that at least one module of this engine
  //                       has reached high time and needs high time repair.
  private double timeToHighTime;
  private boolean reachedHighTime;

  //INSTANCE VARIABLES - Failure Variables
  //     timeToFailure = The least timeToFailure of all this engines modules.
  //     reachedFailure = Indicates that at least one module of this engine
  //                      has reached failure and needs failure repair.
  private double timeToFailure;
  boolean reachedFailure;

  //CONSTRUCTOR METHODS
  public Engine (int srn, Module[] mods) {
    this.serno = srn;
    numberOfModules = EngineBlueprint.getNumberOfModules();
```

```
  if (mods.length == numberOfModules) {
    numberInstalledModules = numberOfModules;
  }
  else {
    throw new IllegalArgumentException("Engine Class Constructor Method - " +
      "insufficient number of modules provided.");
  }
  this.modules = new Module[numberOfModules];
  for (int index = 0; index < numberOfModules; index++){
    if (mods[index].getType() == EngineBlueprint.getModuleTypeOfIndex(index)) {
      this.modules[index] = mods[index];
      modules[index].setEngineIn(this);
    }
    else {
      throw new IllegalArgumentException("Engine Class Constructor Method - " +
        "modules for this engine do not conform to the EngineBlueprint.");
    }
  }
  timeOnEngine = 0.0;
  timeBetweenRepair = 0.0;
  this.updateTimeToHighTime();
  this.updateTimeToFailure();
  this.updateIsUsable();
  if (!isUsable) {
    throw new IllegalArgumentException("Engine Class Constructor Method - " +
      "new engine created but not usable.");
  }
}

//INSTANCE METHODS

//INSTANCE METHODS - to reset the simulation
public void simulationReset(Module[] newModules) {
  for (int index=0; index < numberOfModules; index++){
    modules[index] = newModules[index];
    modules[index].setEngineIn(this);
  }
  numberInstalledModules = numberOfModules;
  timeOnEngine = 0.0;
  timeBetweenRepair = 0.0;
  this.updateTimeToHighTime();
  this.updateTimeToFailure();
  this.updateIsUsable();
  if (!isUsable) {
    throw new IllegalArgumentException("Engine Class simulationReset Method - " +
      "new engine created but not usable.");
  }
}


//INSTANCE METHODS - Getter Methods

public int getSerno() { return serno; }

public int getNumberInstalledModules() { return numberInstalledModules; }
```

```
public double getTimeOnEngine() { return timeOnEngine; }

public double getTimeBetweenRepair() { return timeBetweenRepair; }

public boolean getIsUsable() { return isUsable; }

public double getTimeToHighTime() { return timeToHighTime; }

public boolean getReachedHighTime() { return reachedHighTime; }

public double getTimeToFailure() { return timeToFailure; }

public boolean getReachedFailure() {return reachedFailure; }

public Module getModule(int index) {
  if ((index >= EngineBlueprint.getNumberOfModules()) || (index < 0)){
     throw new IllegalArgumentException("Engine Class getModule Method - " +
       "index out of bounds.");
   }
  else {
    return modules[index];
  }
}

public boolean getIsModuleInstalled(int index) {
   boolean isModuleInstalled = false;
   if ((index >= EngineBlueprint.getNumberOfModules()) || (index < 0)){
     throw new IllegalArgumentException("Engine Class getIsModuleInstalled Method - " +
       "index out of bounds.");
   }
   else {
     if (modules[index] != null) {
       isModuleInstalled = true;
     }
   }
   return isModuleInstalled;
}

//This method should only be called on an engine that does not have all of its
//modules installed, but those installed are usable.  If this engine has no
//modules installed, then value of -1.0 is returned to indicate such.
public double getPartialEngineTimeToHighTime() {
   boolean atLeastOneInstalledModule = false;
   double moduleHighTime = 0.0;
   double partialTimeToHighTime = Double.MAX_VALUE;
   for (int index = 0; index < numberOfModules; index++) {
     if (modules[index] != null) {
       moduleHighTime = modules[index].getClosestHighTime();
       if (moduleHighTime <= 0.0) {
         throw new IllegalArgumentException("Engine Class " +
           "getPartialEngineTimeToHoghTime Method - " +
           "module high time <= 0.  This module should have been removed.");
       }
       if (moduleHighTime < partialTimeToHighTime) {
         atLeastOneInstalledModule = true;
         partialTimeToHighTime = moduleHighTime;
```

```
        }
      }
    }
    if (!atLeastOneInstalledModule) {
      partialTimeToHighTime = -1.0;
    }
    return partialTimeToHighTime;
  }

  //INSTANCE METHODS - Updates to Variables

  public void updateTimeToHighTime() {

    double moduleHighTime = 0.0;
    double storage = Double.MAX_VALUE;

    if (numberInstalledModules == numberOfModules) {
      for (int index = 0; index < numberOfModules; index++) {
        moduleHighTime = modules[index].getClosestHighTime();
        if (moduleHighTime < storage) {
          storage = moduleHighTime;
        }
      }
      timeToHighTime = storage;
      if (timeToHighTime <= 0.0) {
        reachedHighTime = true;
        isUsable = false;
      }
      else {
        reachedHighTime = false;
      }
    }
  }

  public void updateTimeToFailure() {

    double storage = Double.MAX_VALUE;

    if (numberInstalledModules == numberOfModules) {
      for (int index = 0; index < numberOfModules; index++) {
        if (modules[index].getTimeToFailure() < storage) {
          storage = modules[index].getTimeToFailure();
        }
      }
      timeToFailure = storage;
      if (timeToFailure <= 0.0) {
        reachedFailure = true;
        isUsable = false;
      }
      else {
        reachedFailure = false;
      }
    }
  }

  public void updateIsUsable() {
```

```
  if (!reachedHighTime && !reachedFailure &&
    (numberInstalledModules == numberOfModules)) {
      isUsable = true;
  }
  else {
    isUsable = false;
  }
}

public void useEngine(double timeUsed){
  if (timeUsed < 0.0) {
    throw new IllegalArgumentException("Engine Class useEngine Method - " +
      "timeUsed less than zero.");
  }
  if (!isUsable) {
    throw new IllegalArgumentException("Engine Class useEngine Method - " +
      "engine is unusable.");
  }
  if ((timeToHighTime < 0.0) || (reachedHighTime)){
    throw new IllegalArgumentException("Engine Class useEngine Method - " +
      "cannot use an engine that has previously reached high time.");
  }
  if ((timeToFailure < 0.0) || (reachedFailure)){
    throw new IllegalArgumentException("Engine Class useEngine Method - " +
      "cannot use an engine that has previously reached failure.");
  }
  timeOnEngine += timeUsed;
  timeBetweenRepair += timeUsed;
  for (int index = 0; index < numberOfModules; index++) {
    modules[index].useModule(timeUsed);
  }
  this.updateTimeToFailure();
  this.updateTimeToHighTime();
  this.updateIsUsable();
}

//INSTANCE METHODS - Engine Work Methods - i.e. someone is working on the engine.

public Module removeModule(ModuleType typeToRemove) {
  int index = EngineBlueprint.getIndexOfModuleType(typeToRemove);
  return this.removeModuleAtIndex(index);
}

public Module removeModuleAtIndex(int index) {
  if ((index < 0) || (index >= EngineBlueprint.getNumberOfModules())) {
    throw new IllegalArgumentException("Engine Class removeModuleAtIndex Method - " +
      "illegal module index requested.");
  }
  else {
    if (modules[index] == null) {
      throw new IllegalArgumentException("Engine Class removeModuleAtIndex Method - " +
      "no module at that index.");
    }
    else {
      Module removedModule = modules[index];
      removedModule.setEngineIn(null);
```

```
            modules[index] = null;
            numberInstalledModules--;
            isUsable = false;
            return removedModule;
          }
        }
      }

      public void installModule(Module moduleToInstall) {
        if (!moduleToInstall.getIsUsable()) {
          throw new IllegalArgumentException("Engine Class installModule Method - " +
            "Cannot install an unusable module in an engine.");
        }
        boolean installedAModule = false;
        for (int index = 0; index < numberOfModules; index++) {
          if((moduleToInstall.getType() == EngineBlueprint.getModuleTypeOfIndex(index))
            && (modules[index] == null)) {
              modules[index] = moduleToInstall;
              modules[index].setEngineIn(this);
              numberInstalledModules++;
              timeBetweenRepair  = 0.0;
              installedAModule = true;
          }
        }
        if (!installedAModule) {
          throw new IllegalArgumentException("Engine Class removeModule Method - " +
            "Could not install a module because one already installed.");
        }
        else {
          this.updateTimeToHighTime();
          this.updateTimeToFailure();
          this.updateIsUsable();
        }
      }

      public Module swapModule(Module replacementModule){
        if (!replacementModule.getIsUsable()) {
          throw new IllegalArgumentException("Engine Class swapModule Method - " +
            "Cannot install an unusable module in an engine as a swap.");
        }
        Module removedModule = null;
        removedModule = this.removeModule(replacementModule.getType());
        this.installModule(replacementModule);
        return removedModule;
      }
    }
```

APPENDIX H. F18HORNET CLASS JAVA CODE

```
/**
 * LCDR Eric J. Schoch
 * F-18 Thesis Work
 * Last Updated: 11 AUG 03
 * <P>
 * Comments: Simulates the flight schedule and engines of an F-18.
 **/

package jet;

import java.text.DecimalFormat;
import simkit.*;

public class F18Hornet extends SimEntityBase {

  //INSTANCE VARIABLES
  // serno = A serial number specific to this aircraft.
  // engine1 = An engine from the Engine class.
  // engine2 = An engine from the Engine class.
  // isFlyable = A boolean that tells if this aircraft is flyable.
  // flightSchedule = An array that dictates the flight schedule of this
  //                  aircraft.  This schedule goes in pairs.  The first being
  //                  ground time and the second being flight time.  Thus, an
  //                  array of {12,2,24,2} would mean that the plane is on the
  //                  ground 12 hours, in the air 2, on the ground 24, and in the
  //                  air 2.  The aircraft will repeat this schedule.
  // flightSchedulePointer = Keeps tracked of where the aircraft is in its
  //                         flight schedule.
  // weeksDelay = The number of weeks after the start of the simulation that this
  //              aircraft will begin its flight schedule.
  // downTime = For the most recent failure, how long the aircraft has waited
  //            for replacement engines.
  // totalDownTime = Total time this aircraft had to wait for engine - i.e.
  //                 in its lifespan, how many hours total has it been grounded.
  // missedFlightTime = For a specific instance, the amount of flight time the aircraft
  //                    missed because it had to land early due to engine failure.
  // totalMissedFlightTime = Total amount of flight time this aircraft missed because
  //                         it was grounded.
  // totalActualFlightTime = Amount of time aircraft has actually flown.
  // creationTime = Sim time when this aircraft object was created.

  public int serno;
  public Engine engine1;
  public Engine engine2;
  public boolean isFlyable;
  public double[] flightSchedule;
  public int flightSchedulePointer;
  public int weeksDelay;
  public double downTime;
  public double totalDownTime;
  public double missedFlightTime;
  public double totalMissedFlightTime;
```

```
public double totalActualFlightTime;
public double creationTime;
DecimalFormat format0;
DecimalFormat format2;
DecimalFormat format4;


//CONSTRUCTOR METHODS
public F18Hornet(int srn, Engine eng1, Engine eng2, double[] sked, int delay) {
  this.serno = srn;
  if ((eng1==null)||(eng2==null)) {
    throw new IllegalArgumentException("F18Hornet Class Constructor Method - " +
      "Engine provided to build new aircraft was null.");
  }
  else {
    this.engine1 = eng1;
    this.engine2 = eng2;
  }
  if ( (!engine1.getIsUsable()) || (!engine2.getIsUsable()) ) {
    throw new IllegalArgumentException("F18Hornet Class Constructor Method - " +
      "New aircraft built but failed engine provided.");
  }
  else {
    isFlyable = true;
  }
  if ((sked.length < 2) || (sked.length%2 != 0)) {
    throw new IllegalArgumentException("F18Hornet Class Constructor Method - " +
      "New aircraft was provided an improper flight schedule");
  }
  else {
    this.flightSchedule = (double[]) sked.clone();
  }
  for (int index = 0; index < flightSchedule.length; index++) {
    if (flightSchedule[index] < 0) {
      throw new IllegalArgumentException("F18Hornet Class Constructor Method - " +
        "New aircraft provided flight schedule with negative time block.");
    }
  }
  if (delay < 0) {
    throw new IllegalArgumentException("F18Hornet Class Constructor Method - " +
      "Weeks to delay cannot be negative.");
  }
  else {
    this.weeksDelay = delay;
  }

  flightSchedulePointer = 0;
  downTime = 0.0;
  totalDownTime = 0.0;
  missedFlightTime = 0.0;
  totalMissedFlightTime = 0.0;
  totalActualFlightTime = 0.0;
  creationTime = Schedule.getSimTime();
  format0 = new DecimalFormat(" 000 ");
  format2 = new DecimalFormat(" 000000000.00 ");
  format4 = new DecimalFormat(" 0.0000 ");
```

```
}

//INSTANCE METHODS
//INSTANCE METHODS - simulation reset methods
public void reset(){
  super.reset();
  flightSchedulePointer = 0;
  downTime = 0.0;
  totalDownTime = 0.0;
  missedFlightTime = 0.0;
  totalMissedFlightTime = 0.0;
  totalActualFlightTime = 0.0;
  creationTime = Schedule.getSimTime();
}

public void simulationReset(Engine eng1, Engine eng2) {
  engine1 = eng1;
  engine2 = eng2;
  if ( (!engine1.getIsUsable()) || (!engine2.getIsUsable()) ) {
    throw new IllegalArgumentException("F18Hornet Class simulationReset Method - " +
      "New aircraft built but failed engine provided.");
  }
  else {
    isFlyable = true;
  }
}


//INSTANCE METHODS - Getter Methods

public int getSerno() { return serno; }

public boolean getIsFlyable() { return isFlyable; }

public double getTotalDownTime() { return totalDownTime; }

public double getTotalMissedFlightTime() { return totalMissedFlightTime; }

public double getTotalActualFlightTime() { return totalActualFlightTime; }

public double getCreationTime() { return creationTime; }

public double getLifespan() {
  double lifespan = Schedule.getSimTime() - (creationTime + weeksDelay*168.0);
  if (lifespan < 0.0) {
    lifespan = 0.0;
  }
  return lifespan;
}

public double getAvailableTime() {
  double availableTime = 0.0;
  if (this.getLifespan() > 0.0) {
    availableTime = this.getLifespan() - this.getTotalDownTime();
  }
  return availableTime;
```

```
}

public double getAvailability() {
  double storageAvailability = 1.0;
  double storageAvailableTime = this.getAvailableTime();
  double storageLifespan = this.getLifespan();
  if (storageLifespan > 0.0) {
    storageAvailability = this.getAvailableTime()/this.getLifespan();
  }
  return storageAvailability;
}

public int getNumberOfWorkingEngines() {
  int numberOfWorkingEngines = 0;
  if ((engine1 != null)) {
    if (engine1.getIsUsable()) {
      numberOfWorkingEngines++;
    }
  }
  if ((engine2 != null)) {
    if (engine2.getIsUsable()) {
      numberOfWorkingEngines++;
    }
  }
  return numberOfWorkingEngines;
}

public double getTimeToFailure() {
  if (engine1.getTimeToFailure() < engine2.getTimeToFailure()) {
    return engine1.getTimeToFailure();
  }
  else {
    return engine2.getTimeToFailure();
  }
}

//INSTANCE METHODS - MISC
public void updateIsFlyable() {
  if (this.getNumberOfWorkingEngines() == 2) {
    isFlyable = true;
  }
  else {
    isFlyable = false;
  }
}

//   The only way an engine will be allowed to be cannibalized is if there is
//only one working/installed engine on the aircraft.
public Engine cannibalizeEngine() {
  Engine cannibalizedEngine = null;
  if (this.getNumberOfWorkingEngines() == 1) {
    if (engine1 != null) {
      if (engine1.getIsUsable()) {
        cannibalizedEngine = engine1;
        engine1 = null;
      }
```

```
      }
      else if (engine2 != null) {
        if (engine2.getIsUsable()) {
          cannibalizedEngine = engine2;
          engine2 = null;
        }
      }
    }
    if (cannibalizedEngine == null) {
        throw new IllegalArgumentException("F18Hornet Class cannibalizeEngine Method - " +
          "Aircraft did not have ONLY ONE usable engine to cannibalize.");
    }
    else {
      return cannibalizedEngine;
    }
  }

  public void updateFlightSchedulePointer() {
    if (flightSchedulePointer == (flightSchedule.length - 1) ){
      flightSchedulePointer = 0;
    }
    else {
      flightSchedulePointer++;
    }
  }

  public void flyAircraft(double timeFlown) {
    if (timeFlown <= 0.0) {
        throw new IllegalArgumentException("F18Hornet Class flyAircraft Method - " +
          "Cannot fly negative or zero hours.");
    }
    if (!isFlyable) {
        throw new IllegalArgumentException("F18Hornet Class flyAircraft Method - " +
          "Cannot fly an unflyable aircraft.");
    }
    totalActualFlightTime += timeFlown;
    engine1.useEngine(timeFlown);
    engine2.useEngine(timeFlown);
    this.updateIsFlyable();
    this.updateFlightSchedulePointer();
  }

  //The only reason that this method will be called is if another method knows
  //that at least one of the two currently installed engines has failed and needs
  //removed and shipped.  Thus, if this method is called and the aircraft is
  //still functional, then a error in logic is flagged.
  public void informOLofBadEngines() {
    if( (engine1==null) || (engine2==null) ) {
      throw new IllegalArgumentException("F18Hornet Class removeBadEngines Method - " +
        "Two engines must be installed to call this method.");
    }
    if( (!engine1.getIsUsable()) && (!engine2.getIsUsable()) ) {
      waitDelay ("F18Needs2EnginesRemovedByOL", 0.0, new Object[] {engine1, engine2, this} );
      engine1 = null;
      engine2 = null;
      isFlyable = false;
```

```
    }
    else if(!engine1.getIsUsable()) {
      waitDelay ("F18Needs1EngineRemovedByOL", 0.0, new Object[] {engine1, this} );
      engine1 = null;
      isFlyable = false;
    }
    else if(!engine2.getIsUsable()) {
      waitDelay ("F18Needs1EngineRemovedByOL", 0.0, new Object[] {engine2, this} );
      engine2 = null;
      isFlyable = false;
    }
    if(isFlyable){
      throw new IllegalArgumentException("F18Hornet Class checkEngines Method - " +
        "No engine was removed when at least one should have been.");
    }
  }

  //INSTANCE METHODS - to run the plane's schedule

  public void doRun() {
    waitDelay("ScheduleFlight", weeksDelay*168.0);
  }

  public void doScheduleFlight() {
    waitDelay("TakeOff", flightSchedule[flightSchedulePointer]);
    this.updateFlightSchedulePointer();
  }

  public void doTakeOff() {
    if(flightSchedule[flightSchedulePointer] == 0.0) {
      this.updateFlightSchedulePointer();
      waitDelay("ScheduleFlight",0.0);
    }
    else {
      if(this.getTimeToFailure() <= flightSchedule[flightSchedulePointer] ) {
        waitDelay("LandWithEngineFailure", this.getTimeToFailure());
      }
      else {
        waitDelay("NormalLanding",flightSchedule[flightSchedulePointer]);
      }
    }
  }

  public void doLandWithEngineFailure() {
    missedFlightTime = flightSchedule[flightSchedulePointer] - this.getTimeToFailure() - 0.0000001;
    totalMissedFlightTime += missedFlightTime;
    this.flyAircraft(this.getTimeToFailure() + 0.0000001);
    this.informOLofBadEngines();
  }

  public void doNormalLanding() {
    this.flyAircraft(flightSchedule[flightSchedulePointer]);
    waitDelay("EngineHighTimeCheck",0.0);
  }

  public void doEngineHighTimeCheck() {
```

```
  if (isFlyable){
    waitDelay("ScheduleFlight",0.0);
  }
  else {
    this.informOLofBadEngines();
  }
}

//  The next 2 methods do nothing to the Engine they are passed.  The O Level
// will be listening to this aircraft and will use the Engine that is passed.
public void doF18Needs1EngineRemovedByOL(Engine eng, F18Hornet whoFrom) {
  downTime = Schedule.getSimTime();
}
public void doF18Needs2EnginesRemovedByOL(Engine eng1, Engine eng2, F18Hornet whoFrom) {
  downTime = Schedule.getSimTime();
}

public void doF18EngineInstalledByOL(Engine eng, F18Hornet whoTo) {
  if (whoTo == this) {
    if ((engine1 != null) && (engine2 != null)) {
      throw new IllegalArgumentException("F18Hornet Class doOLtoAEngineMove Method - " +
        "Cannot install an engine on an aircraft with 2 engines already installed.");
    }
    if (!eng.getIsUsable()) {
      throw new IllegalArgumentException("F18Hornet Class doOLtoAEngineMove Method - " +
        "Cannot install a bad engine on an aircraft.");
    }
    if (engine1 == null) {
      engine1 = eng;
    }
    else {
      engine2 = eng;
    }
    this.updateIsFlyable();
    if (isFlyable) {
      waitDelay("F18HasBeenRepaired", 0.0);
    }
    else {
      waitDelay("F18WaitingForSecondEngine",0.0);
    }
  }
}

public void doF18HasBeenRepaired() {
  downTime = Schedule.getSimTime() - downTime;
  totalDownTime += downTime;
  downTime -= missedFlightTime;
  while (downTime > 0.0) {
    downTime -= flightSchedule[flightSchedulePointer];
    if (flightSchedulePointer%2 != 0) {
      totalMissedFlightTime += flightSchedule[flightSchedulePointer];
    }
    this.updateFlightSchedulePointer();
  }
  if (flightSchedulePointer%2 == 0) {
    waitDelay("ScheduleFlight",-1.0*downTime);
```

```
  }
  else {
    waitDelay("TakeOff",-1.0*downTime);
  }
}

public void doF18WaitingForSecondEngine() {
}

//INSTANCE METHODS - reports

//Outputs to screen and returns aircraft availability data to include total
//mission hours missed, total mission hours flow, total down time, total up
//time, and total lifespan.
public double[] reportAircraftAvailabilityScreenWithReturn(){

  double msnHrsMsd = totalMissedFlightTime;
  double msnHrsFln = totalActualFlightTime;
  double timeUnavl = totalDownTime;
  double totalTime = this.getLifespan();



  double returnArray[] = new double[5];

  double availability = 1.0;

  if (!isFlyable) {
    int tempFSPointer = flightSchedulePointer;
    double tempDownTime = Schedule.getSimTime() - downTime;
    timeUnavl += tempDownTime;
    tempDownTime -= missedFlightTime;
    while (tempDownTime > 0.0) {
      tempDownTime -= flightSchedule[tempFSPointer];
      if (tempFSPointer%2 != 0) {
        msnHrsMsd += flightSchedule[tempFSPointer];
      }
      if (tempFSPointer == (flightSchedule.length - 1) ){
        tempFSPointer = 0;
      }
      else {
        tempFSPointer++;
      }
    }
    if (tempFSPointer%2 == 0) {
      msnHrsMsd += (-1.0*tempDownTime);
    }
  }

  if (totalTime > 0.0) {
    availability = (totalTime - timeUnavl)/totalTime;
  }
  else {
    availability = 1.0;
  }
  System.out.println( format0.format(serno) + "  " +
```

```
      format2.format(msnHrsMsd) + " " +
      format2.format(msnHrsFln) +
      format2.format(timeUnavl) +
      format2.format(totalTime - timeUnavl) + " " +
      format2.format(totalTime) + "    " +
      format4.format(availability) );

    returnArray[0] = msnHrsMsd;
    returnArray[1] = msnHrsFln;
    returnArray[2] = timeUnavl;
    returnArray[3] = totalTime - timeUnavl;
    returnArray[4] = totalTime;

    return returnArray;

  }

  //Reurns mission and total availability data to the screen only.
  public void reportAircraftAvailabilityScreenOnly(){

    double msnHrsMsd = totalMissedFlightTime;
    double msnHrsFln = totalActualFlightTime;
    double timeUnavl = totalDownTime;
    double totalTime = this.getLifespan();

    double returnArray[] = new double[5];

    double availability = 1.0;

    if (!isFlyable) {
      int tempFSPointer = flightSchedulePointer;
      double tempDownTime = Schedule.getSimTime() - downTime;
      timeUnavl += tempDownTime;
      tempDownTime -= missedFlightTime;
      while (tempDownTime > 0.0) {
        tempDownTime -= flightSchedule[tempFSPointer];
        if (tempFSPointer%2 != 0) {
          msnHrsMsd += flightSchedule[tempFSPointer];
        }
        if (tempFSPointer == (flightSchedule.length - 1) ){
          tempFSPointer = 0;
        }
        else {
          tempFSPointer++;
        }
      }
      if (tempFSPointer%2 == 0) {
        msnHrsMsd += (-1.0*tempDownTime);
      }
    }

    if (totalTime > 0.0) {
      availability = (totalTime - timeUnavl)/totalTime;
    }
    else {
      availability = 1.0;
```

```
    }
    System.out.println( format0.format(serno) + "  " +
      format2.format(msnHrsMsd) + " " +
      format2.format(msnHrsFln) +
      format2.format(timeUnavl) +
      format2.format(totalTime - timeUnavl) + " " +
      format2.format(totalTime) + "    " +
      format4.format(availability) );

  }

  public double[] reportAircraftAvailabilityReturnOnly(){

    double msnHrsMsd = totalMissedFlightTime;
    double msnHrsFln = totalActualFlightTime;
    double timeUnavl = totalDownTime;
    double totalTime = this.getLifespan();

    double returnArray[] = new double[5];

    double availability = 1.0;

    if (!isFlyable) {
      int tempFSPointer = flightSchedulePointer;
      double tempDownTime = Schedule.getSimTime() - downTime;
      timeUnavl += tempDownTime;
      tempDownTime -= missedFlightTime;
      while (tempDownTime > 0.0) {
        tempDownTime -= flightSchedule[tempFSPointer];
        if (tempFSPointer%2 != 0) {
          msnHrsMsd += flightSchedule[tempFSPointer];
        }
        if (tempFSPointer == (flightSchedule.length - 1) ){
          tempFSPointer = 0;
        }
        else {
          tempFSPointer++;
        }
      }
      if (tempFSPointer%2 == 0) {
        msnHrsMsd += (-1.0*tempDownTime);
      }
    }

    returnArray[0] = msnHrsMsd;
    returnArray[1] = msnHrsFln;
    returnArray[2] = timeUnavl;
    returnArray[3] = totalTime - timeUnavl;
    returnArray[4] = totalTime;

    return returnArray;

  }

}
```

APPENDIX I. FLIGHTSCHEDULE CLASS JAVA CODE

```
/**
 * LCDR Eric J. Schoch
 * F-18 Thesis Work
 * Last Updated: 09 AUG 03
 * Comments:  A class that has various preset flight schedules that may be used
 * in the F18Hornet class.  The flight schedules are array.  The first number
 * represents the ground time and the second the flight time.  The arrays are set
 * up to represent a week of flying.  The week starts at Sunday 0000 hours.
 */

package jet;

import simkit.random.*;

public class FlightSchedules {

  //CLASS VARIABLES
  //The following are custom flight schedules.  Currently, only one is present as
  //an example.
  private static final double[][] masterFlightSchedules = new double[][] {
    new double[] {32.0, 2.0, 22.0, 2.0, 22,0, 2.0, 22.0, 2.0, 22.0, 2.0, 38.0, 0.0} //MTWThF 8to10
  };

  //INSTANCE VARIABLES
  RandomVariate uniformRV;

  //CONSTUCTOR METHOD
  public FlightSchedules(RandomVariate rv) {
    this.uniformRV = rv;
  }

  //INSTANCE METHODS

  //INSTANCE METHODS - Five Day Block Methods
  private double[] getFiveDayBlock (double startDayHour, double startTime, double flightTime) {
    double flightSchedule[] = new double[12];
    flightSchedule[0] = startDayHour + startTime;
    flightSchedule[10] = 168.0 - (startDayHour+startTime) - (5*flightTime) - (4*(24.0-flightTime));
    flightSchedule[11] = 0.0;
    for (int index = 0; index < 5; index++) {
      flightSchedule[1 + index*2] = flightTime;
    }
    for (int index = 0; index<4; index++) {
      flightSchedule[2 + index*2] = 24.00 - flightTime;
    }
    return flightSchedule;
  }

  public double[] getSMTWTh(double start, double flightTime) {
    return this.getFiveDayBlock(0.0, start, flightTime);
  }
```

```
public double[] getMTWThF(double start, double flightTime) {
  return this.getFiveDayBlock(24.0, start, flightTime);
}

public double[] getTWThFS(double start, double flightTime) {
  return this.getFiveDayBlock(48.0, start, flightTime);
}

public double[] getRandomFiveDayBlock(double flightTime) {
  double randomDay = uniformRV.generate();
  double randomStartTime = (uniformRV.generate())*(24.0 - flightTime);
  double[] returnArray = null;
  if (randomDay <= (1.0/3.0)) {
    returnArray = this.getSMTWTh(randomStartTime, flightTime);
  }
  else if (randomDay <= (2.0/3.0)) {
    returnArray = this.getMTWThF(randomStartTime, flightTime);
  }
  else if (randomDay <= 1.0) {
    returnArray = this.getTWThFS(randomStartTime, flightTime);
  }
  return returnArray;
}

//INSTANCE METHODS - Three Day Block Methods
public double[] getThreeDayBlock (double startDayHour, double startTime, double flightTime) {
  double flightSchedule[] = new double[8];
  flightSchedule[0] = startDayHour + startTime;
  flightSchedule[6] = 168.0 - (startDayHour+startTime) - (3*flightTime) - (2*(24.0-flightTime));
  flightSchedule[7] = 0.0;
  for (int index = 0; index < 3; index++) {
    flightSchedule[1 + index*2] = flightTime;
  }
  for (int index = 0; index<2; index++) {
    flightSchedule[2 + index*2] = 24.00 - flightTime;
  }
  return flightSchedule;
}

public double[] getSMT(double start, double flightTime) {
  return this.getThreeDayBlock(0.0, start, flightTime);
}

public double[] getMTW(double start, double flightTime) {
  return this.getThreeDayBlock(24.0, start, flightTime);
}

public double[] getTWTh(double start, double flightTime) {
  return this.getThreeDayBlock(48.0, start, flightTime);
}

public double[] getWThF(double start, double flightTime) {
  return this.getThreeDayBlock(72.0, start, flightTime);
}

public double[] getThFS(double start, double flightTime) {
```

```
    return this.getThreeDayBlock(96.0, start, flightTime);
  }

  public double[] getRandomThreeDayBlock(double flightTime) {
    double randomDay = uniformRV.generate();
    double randomStartTime = (uniformRV.generate())*(24.0 - flightTime);
    double[] returnArray = new double[8];
    if (randomDay <= 0.2) {
      returnArray = this.getSMT(randomStartTime, flightTime);
    }
    else if (randomDay <= 0.4) {
      returnArray = this.getMTW(randomStartTime, flightTime);
    }
    else if (randomDay <= 0.6) {
      returnArray = this.getTWTh(randomStartTime, flightTime);
    }
    else if (randomDay <= 0.8) {
      returnArray = this.getWThF(randomStartTime, flightTime);
    }
    else if (randomDay <= 1.0) {
      returnArray = this.getThFS(randomStartTime, flightTime);
    }
    return returnArray;
  }

  public double[] getCustomSchedule(int index) {
    return masterFlightSchedules[index];
  }

}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX J. OLEVEL CLASS JAVA CODE

```
/**
 * LCDR Eric J. Schoch
 * F-18 Thesis Work
 * Last Updated: 21JUL03
 * <P>
 * Comments:  The Organizational Level of the simulation.
 **/

package jet;
import simkit.*;
import java.util.*;

public class OLevel extends SimEntityBase {

  //INSTANCE VARIABLES
  // uic = the unit identification code - i.e. who this is
  // cannibalize = indicates if thos OLevel will take two aircraft with only
  //             one working engine each and combine the good engines into
  //             one of the two aircraft so at least one may fly.
  // numberOfCannibalizations = keeps tracked of how many times it has happened
  // engineTransferTime = the shipping time the OLevel to the supporting ILevel
  //                   (also the time from the ILevel to the OLevel)
  // needs1Engine = keeps tracked of the aircraft that are waiting on one engine
  //              from the ILevel.
  // needs2Engines = same as 1 but aircraft needs 2 good engines.
  // aircraftSupported = a LinkedList of what aircraft this OLevel supports.
  public int uic;
  public boolean cannibalize;
  public int numberOfCannibalizations;
  public double engineTransferTime;
  public LinkedList needs1Engine;
  public LinkedList needs2Engines;
  public LinkedList aircraftSupported;

  //CONSTRUCTOR METHODS
  public OLevel(int name, boolean can, double et) {
    this.uic = name;
    this.cannibalize = can;
    numberOfCannibalizations = 0;
    this.engineTransferTime = et;
    needs1Engine = new LinkedList();
    needs2Engines = new LinkedList();
    aircraftSupported = new LinkedList();
  }

  //INSTANCE METHODS
  //INSTANCE METHODS - to reset this SimEntityBase object
  public void reset() {
    super.reset();
    numberOfCannibalizations = 0;
    needs1Engine.clear();
    needs2Engines.clear();
```

```
        }

        //INSTANCE METHODS - to schedule to O Level
        public void doF18Needs1EngineRemovedByOL(Engine badEngine, F18Hornet whoFrom) {
          waitDelay("OLComplete1EngineInspection",
EngineBlueprint.getEngineOLTroubleshootTime(),
            new Object[] {badEngine, whoFrom} );
        }

        public void doOLComplete1EngineInspection(Engine badEngine, F18Hornet whoFrom) {
          waitDelay("OLComplete1EngineRemoval", EngineBlueprint.getEngineOLRemovalTime(),
            new Object[] {badEngine, whoFrom} );
        }

        public void doOLComplete1EngineRemoval(Engine badEngine, F18Hornet whoFrom) {
          needs1Engine.add(whoFrom);
          waitDelay("OLRequestsEngineFromIL", 0.0, new Object[] {this, new Integer(1)});
          waitDelay("StartOLtoILBadEngineMove", 0.0, badEngine);
        }

        public void doF18Needs2EnginesRemovedByOL(Engine badEngine1, Engine badEngine2,
F18Hornet whoFrom) {

waitDelay("OLComplete2EnginesInspection",EngineBlueprint.getEngineOLTroubleshootTime(),
            new Object[] {badEngine1, badEngine2, whoFrom} );
        }

        public void doOLComplete2EnginesInspection(Engine badEngine1, Engine badEngine2,
F18Hornet whoFrom) {
          waitDelay("OLComplete2EnginesRemoval", EngineBlueprint.getEngineOLRemovalTime(),
            new Object[] {badEngine1, badEngine2, whoFrom} );
        }

        public void doOLComplete2EnginesRemoval(Engine badEngine1, Engine badEngine2,
F18Hornet whoFrom) {
          needs2Engines.add(whoFrom);
          waitDelay("OLRequestsEngineFromIL", 0.0, new Object[] {this, new Integer(2)});
          waitDelay("OLRequestsEngineFromIL", 0.0, new Object[] {this, new Integer(2)});
          waitDelay("StartOLtoILBadEngineMove", 0.0, badEngine1);
          waitDelay("StartOLtoILBadEngineMove", 0.0, badEngine2);
        }

        public void doStartOLtoILBadEngineMove(Engine badEngine) {
          waitDelay("CompleteOLtoILBadEngineMove", engineTransferTime, badEngine);

        }

        // The following two methods are just placeholder methods (they do nothing here,
        // but do something in the ILevel Class).
        public void doOLRequestsEngineFromIL(OLevel whoIsChecking, int numberOfFails) {
        }
        public void doCompleteOLtoILBadEngineMove(Engine badEngine) {
        }

        public void doStartILtoOLEngineMove(Engine goodEngine, OLevel whoTo) {
          if (whoTo == this) {
```

```
            waitDelay("CompleteILtoOLEngineMove", engineTransferTime, goodEngine);
          }
        }

        public void doCompleteILtoOLEngineMove(Engine goodEngine){
          waitDelay("StartOLEngineInstall", 0.0, goodEngine);
        }

        public void doStartOLEngineInstall(Engine goodEngine){
          waitDelay("CompleteOLEngineInstall", EngineBlueprint.getEngineOLInstallTime(),
goodEngine);
        }

        public void doCompleteOLEngineInstall(Engine goodEngine) {
          if (needs1Engine.size() == 0) {
            needs1Engine.add((F18Hornet) needs2Engines.getFirst());
            waitDelay("F18EngineInstalledByOL", 0.0,
              new Object[] {goodEngine, (F18Hornet) needs2Engines.removeFirst()} );
          }
          else {
            waitDelay("F18EngineInstalledByOL", 0.0,
              new Object[] {goodEngine, (F18Hornet) needs1Engine.removeFirst()} );
          }
        }

        public void doOLTryToCannibalize(OLevel whoShouldTry) {
          if (cannibalize) {
            if (whoShouldTry == this) {
              if (needs1Engine.size()>1) {
                waitDelay("StartEngineCannibalizationAtOL", 0.0);
              }
            }
          }
        }

        public void doStartEngineCannibalizationAtOL() {
          numberOfCannibalizations++;
          Engine removedEngine = ((F18Hornet) needs1Engine.getLast()).cannibalizeEngine();
          needs2Engines.add((F18Hornet) needs1Engine.removeLast());
          F18Hornet whoTo = (F18Hornet) needs1Engine.removeFirst();
          //Time below is 0.0 because when this method is called, the time to remove
          //an engine has already been added.
          waitDelay("CannibalizedEngineRemoved", 0.0, new Object[] { removedEngine, whoTo} );
        }

        public void doCannibalizedEngineRemoved(Engine removedEngine, F18Hornet whoTo){
          waitDelay("F18EngineInstalledByOL", EngineBlueprint.getEngineOLInstallTime(),
            new Object[] { removedEngine, whoTo} );
        }

        // The following method is just a placeholder method (it does nothing here,
        // but do something in the F18Hornet Class).
        public void doF18EngineInstalledByOL(Engine goodEngine, F18Hornet whoTo) {
        }

        //INSTANCE METHODS - to track and report on the aircraft supported
```

```
  public void addAircraftSupported(F18Hornet aircraft) {
    aircraftSupported.add(aircraft);
  }

  public void removeAircraftSupported(F18Hornet aircraft) {
    aircraftSupported.remove(aircraft);
  }

  public double reportOLevelAvailability() {
    double availability = 0.0;
    F18Hornet aircraft = null;
    for (int index=0; index < aircraftSupported.size(); index++) {
      aircraft = (F18Hornet) aircraftSupported.get(index);
      availability += aircraft.getAvailability();
    }
    return availability;
  }

  public void onscreenOLevelAvailability() {
    F18Hornet aircraft = null;
    System.out.println("OLevel UIC : " + uic);
    System.out.println("Availability by aircraft");
    for (int index=0; index < aircraftSupported.size(); index++) {
      aircraft = (F18Hornet) aircraftSupported.get(index);
      System.out.println("   Serno : " + aircraft.getSerno() + " Ao : " +
        aircraft.getAvailability());
      System.out.println("TOTAL OLevel Ao : " + this.reportOLevelAvailability());
    }
  }

  public void onscreenAircraftSupported() {
    F18Hornet aircraft = null;
    System.out.println("OLevel UIC : " + uic + " Total Aircraft Supported : " +
        aircraftSupported.size());
    for (int index=0; index < aircraftSupported.size(); index++) {
      aircraft = (F18Hornet) aircraftSupported.get(index);
      System.out.println("   Aircraft #" + index + " Serno : " + aircraft.getSerno());
    }
  }

  public int getNumberOfCannibalizations() { return numberOfCannibalizations; }

}
```

APPENDIX K. ILEVEL CLASS JAVA CODE

```
/**
 * LCDR Eric J. Schoch
 * F-18 Thesis Work
 * Last Updated: 11 AUG 03
 * <P>
 * Comments: A class to simulated the I-Level repair process of F-18 engines.
 * THIS I LEVEL INCORPORATES THE I3 TO D REPAIR PHILOSOPHY AND THUS IS NOT
 * DESIGNED TO HANDLE MODULE REPAIR AT THE I LEVEL.
 **/

package jet;
import java.util.*;
import simkit.*;
import java.text.DecimalFormat;
import simkit.random.*;

public class ILevel extends SimEntityBase {

  //INSTANCE VARIABLE
  // uic = The unit identification code of this I Level.
  // iLtoDLTransferTime = How long to ship a module from the I Level to the D Level.
  // engineAllowance = Stock allowance for engines.
  // moduleAllowance = An array of module allowances.
  // engineNIS = Keeps tracked of the number of engine Not In Stock experienced.
  // moduleNIS = Keeps tracked of the number of module Not in STock experienced
  //           by module type;
  // needsEngine = LinkedList of those OLevel activities that need an engine.
  // goodEnginePool = I Level's pool of good engines available for transfer.
  // badEnginePool = I Level's pool of those engines waiting parts (modules).
  // goodModuleList = A LinkedList array of all good modules at this ILevel.  This
  //               includes all good modules installed in engines awaiting parts
  //               and all good modules not installed in an engine.
  // oLevelsSupported = A LinkedList of all the O Levels this I Level supports.
  // numberOfModules = How many modules EngineBlueprint says are in every engine;
  int uic;
  public double iLtoDLTransferTime;
  public int engineAllowance;
  public int[] moduleAllowance;
  public int engineNIS;
  public int[] moduleNIS;
  public LinkedList needsEngine;
  public LinkedList goodEnginePool;
  public LinkedList badEnginePool;
  public LinkedList[] goodModuleList;
  public LinkedList oLevelSupported;
  public int numberOfModules;

  DecimalFormat format0;


  //CONSTRUCTOR METHODS
  public ILevel(int name, double xfer, int engAll, int[] modAll) {
```

```
  this.uic = name;
  numberOfModules = EngineBlueprint.getNumberOfModules();
  if (xfer < 0.0) {
    throw new IllegalArgumentException("ILevel Class Constructor Method - " +
      "iLtoDLTransferTime cannot be less than zero.");
  }
  else {
    iLtoDLTransferTime = xfer;
  }
  if (engAll < 0) {
    throw new IllegalArgumentException("ILevel Class Constructor Method - " +
      "engine allowance cannot be less than zero.");
  }
  else {
    engineAllowance = engAll;
  }
  if (modAll.length != numberOfModules) {
    throw new IllegalArgumentException("ILevel Class Constructor Method - " +
      "moduleAllowance does not contain correct number of allowances.");
  }
  else {
    moduleAllowance = new int[numberOfModules];
    moduleNIS = new int[numberOfModules];
  }
  for (int index = 0; index < numberOfModules; index++){
    if (modAll[index] < 0) {
      throw new IllegalArgumentException("ILevel Class Constructor Method - " +
        "moduleAllowance can not be less than zero.");
    }
    else {
      moduleAllowance[index] = modAll[index];
      moduleNIS[index] = 0;
    }
  }
  engineNIS = 0;
  goodModuleList = new LinkedList[numberOfModules];
  for (int index = 0; index < numberOfModules; index++) {
    goodModuleList[index] = new LinkedList();
  }
  needsEngine = new LinkedList();
  goodEnginePool = new LinkedList();
  badEnginePool = new LinkedList();
  oLevelSupported = new LinkedList();

  format0 = new DecimalFormat(" 000 ");

}

//INSTANCE METHODS
//INSTANCE METHODS - Simulation Reset
public void reset() {
  engineNIS = 0;
  for (int index = 0; index < numberOfModules; index++) {
    moduleNIS[index] = 0;
  }
  needsEngine.clear();
```

```
  badEnginePool.clear();
}

public void simulationReset() {
  goodEnginePool.clear();
  for (int index = 0; index < numberOfModules; index++) {
    goodModuleList[index].clear();
  }
}

//INSTANCE METHODS - Getter Methods
public int getEngineAllowance() { return engineAllowance; };

public int[] getModuleAllowance() { return moduleAllowance; }

public int getSpecificModuleAllowance(int moduleIndex) {
  return moduleAllowance[moduleIndex];
}

public int getEngineNIS() { return engineNIS; }

public int[] getModuleNIS() { return moduleNIS; }

//INSTANCE METHODS - to schedule the I Level

public void doOLRequestsEngineFromIL(OLevel whoNeedsIt, int engineFailsThisInstance) {
  if (goodEnginePool.size() > 0) {
    waitDelay("OLRequestedEngineISAtIL", 0.0,
      new Object[] {goodEnginePool.removeFirst(), whoNeedsIt});
  }
  else{
    waitDelay("OLRequestedEngineNISAtIL", 0.0,
      new Object[] {whoNeedsIt, new Integer(engineFailsThisInstance)} );
  }
}

public void doOLRequestedEngineISAtIL(Engine engineToGive, OLevel whoNeedsIt){
  waitDelay("StartILtoOLEngineMove", 0.0, new Object[] {engineToGive, whoNeedsIt});
}

public void doOLRequestedEngineNISAtIL(OLevel whoNeedsIt, int engineFailsThisInstance){
  engineNIS++;
  needsEngine.add(whoNeedsIt);
  if (engineFailsThisInstance == 1) {
    waitDelay("OLTryToCannibalize", 0.0, whoNeedsIt);
  }
}

public void doCompleteOLtoILBadEngineMove(Engine badEngine) {
  waitDelay("StartILBadEngineInspection",0.0, badEngine);
}

public void doStartILBadEngineInspection(Engine badEngine) {
  waitDelay("CompleteILBadEngineInspection",
    EngineBlueprint.getEngineILInspectionTime(), badEngine);
}
```

```
public void doCompleteILBadEngineInspection(Engine badEngine){
  waitDelay("ILDistributeEngineAndModules", 0.0, badEngine);
}

public void doILDistributeEngineAndModules(Engine badEngine){
  Module storeMod = null;
  for (int index = 0; index < numberOfModules; index++) {
    storeMod = badEngine.getModule(index);
    if ( (!storeMod.getIsUsable()) ||
      (storeMod.getClosestHighTime() < EngineBlueprint.getBuildWindow(index)) ||
      (storeMod.getTimeToFailure() < EngineBlueprint.getFailureDependencyTime(index)) ) {
        waitDelay("StartILBadModuleRemoval", 0.0, badEngine.removeModuleAtIndex(index));
    }
    else {
      waitDelay("ILAddGoodModuleToAvailableList",0.0, storeMod);
    }
  }
  waitDelay("ILAddAWPEngineToAWPEnginePool", 0.0, badEngine);
}

public void doStartILBadModuleRemoval(Module badModule){
  waitDelay("CompleteILBadModuleRemoval",
    EngineBlueprint.getModuleRemovalTime(badModule.getModuleIndex()), badModule);
}

public void doCompleteILBadModuleRemoval(Module badModule){
  waitDelay("StartILtoDLBadModuleMove", 0.0, badModule);
  waitDelay("ILRequestsModuleFromDL", 0.0, new Object[]{badModule, this});
}

public void doStartILtoDLBadModuleMove(Module badModule){
  waitDelay("CompleteILtoDLBadModuleMove", iLtoDLTransferTime, badModule);
}

public void doILAddGoodModuleToAvailableList(Module goodModule){
  this.addGoodModuleToPool(goodModule);
}

public void doILAddAWPEngineToAWPEnginePool(Engine badEngine){
  badEnginePool.add(badEngine);
  waitDelay("ILTryToBuildAnEngine", 0.01);
}

public void doILTryToBuildAnEngine() {
  if (badEnginePool.size()>0) {
    boolean canBuild = true;
    for (int index=0; index < numberOfModules; index++) {
      if (goodModuleList[index].size() == 0){
        canBuild = false;
        moduleNIS[index]++;
      }
    }
    if (canBuild) {
      // This method examines each engine in the badEnginePool and the modules to
      // repair it from all other installed (in other engines) and uninstalled
```

```
        // modules.  From this, it produces the engine with the highest time to high
        // time.  Cannibalization is allowed.
        Engine bestEngine = null;
        Engine storageEngine = null;
        Module[] bestModules = new Module[numberOfModules];
        Module[] storageModules = new Module[numberOfModules];
        double bestHighTime = 0.0;
        double highTimeWhenConstructed = Double.MAX_VALUE;
        double timeToBuildEngine = 0.0;

        //Find the best engine to build.
        for (int index1 = 0; index1<badEnginePool.size(); index1++) {
          storageEngine = (Engine)badEnginePool.get(index1);
          highTimeWhenConstructed = storageEngine.getPartialEngineTimeToHighTime();
          if (highTimeWhenConstructed < 0) {
            highTimeWhenConstructed = Double.MAX_VALUE;
          }
          for (int index = 0; index<numberOfModules; index++) {
            storageModules[index] = null;
          }
          for (int index2 = 0; index2<numberOfModules; index2++) {
            if(!storageEngine.getIsModuleInstalled(index2)){
              storageModules[index2] = this.findClosestHighTime(highTimeWhenConstructed, index2);
              if(storageModules[index2].getClosestHighTime() < highTimeWhenConstructed) {
                highTimeWhenConstructed = storageModules[index2].getClosestHighTime();
              }
            }
          }
          if (highTimeWhenConstructed > bestHighTime) {
            bestHighTime = highTimeWhenConstructed;
            bestEngine = storageEngine;
            for (int index = 0; index<numberOfModules; index++) {
              bestModules[index] = storageModules[index];
            }
          }
        }

        //Build the best engine found above.
        for (int index = 0; index<numberOfModules; index++) {
          if ((!bestEngine.getIsModuleInstalled(index)) && (bestModules[index] != null)){
            if (bestModules[index].getEngineIn() != null) {

bestEngine.installModule((bestModules[index].getEngineIn()).removeModuleAtIndex(index));
              timeToBuildEngine += EngineBlueprint.getModuleRemovalTime(index) +
                EngineBlueprint.getModuleInstallTime(index);
            }
            else {
              bestEngine.installModule(bestModules[index]);
              timeToBuildEngine += EngineBlueprint.getModuleInstallTime(index);
            }
          }
          else if (bestEngine.getIsModuleInstalled(index) && (bestModules[index] != null)){
            throw new IllegalArgumentException ("ILevel Class doBuildEngine Method - " +
              "this engine already has a working module of the type provided.");
          }
          else if ( (!(bestEngine.getIsModuleInstalled(index)) && (bestModules[index] == null) )){
```

```
            throw new IllegalArgumentException ("ILevel Class doBuildEngine Method - " +
              "a module was not provided to fill a hole in this engine.");
          }
        }
        if (!bestEngine.getIsUsable()) {
          throw new IllegalArgumentException ("ILevel Class doBuildEngine Method - " +
            "unusable engine built but thought to be usable by I Level.");
        }
        else {
          for (int index = 0; index < numberOfModules; index++) {
            goodModuleList[index].remove(bestEngine.getModule(index));
          }
          badEnginePool.remove(bestEngine);
          //The above calculates timeToBuildEngine based on the sum of the parts
          //removed and installed.  The next line assumes one time to build an
          //engine reguardless of the scope of work invloved.  Comment this line
          //out if you desire to revert to the sum of parts calculation.
          timeToBuildEngine = EngineBlueprint.getEngineBuildUpTime();
          waitDelay("CompleteILBuildEngine", timeToBuildEngine, bestEngine);
        }
      }

      else{
        waitDelay("ILNotEnoughModulesToBuildAnEngine",0.0);
      }
    }
  }


public void doCompleteILBuildEngine(Engine goodEngine) {
  if (needsEngine.size() > 0) {
    waitDelay("ILRepairedEngineNeededAtOL", 0.0,
      new Object[] {goodEngine, needsEngine.removeFirst()});
  }
  else{
    waitDelay("ILAddEngineToGoodEnginePool", 0.0, goodEngine);
  }
}

public void doILRepairedEngineNeededAtOL(Engine goodEngine, OLevel whoTo){
  waitDelay("StartILtoOLEngineMove", 0.0, new Object[]{goodEngine, whoTo} );
}

public void doILAddEngineToGoodEnginePool(Engine goodEngine) {
  goodEnginePool.add(goodEngine);
}

public void doStartDLtoILModuleMove(Module goodModule, ILevel whoTo) {
  if (whoTo == this) {
    waitDelay("CompleteDLtoILModuleMove", iLtoDLTransferTime, goodModule);
  }
}

public void doCompleteDLtoILModuleMove(Module goodModule) {
  waitDelay("ILAddRepairedModuleToPool", 0.0, goodModule);
}
```

```
  public void doILAddRepairedModuleToPool(Module goodModule) {
    this.addGoodModuleToPool(goodModule);
    waitDelay("ILTryToBuildAnEngine",0.0);
  }



  //The next methods are just "place holders" - i.e. used in another class, not here.
  public void doStartILtoOLEngineMove(Engine goodEngine, OLevel whoTo) {
  }
  public void doOLTryToCannibalize(OLevel whoShouldTry) {
  }
  public void doILRequestsModuleFromDL(Module badModule){
  }
  public void doCompleteILtoDLBadModuleMove(Module badModule){
  }
  public void doILNotEnoughModulesToBuildAnEngine() {
  }

  //INSTANCE METHODS - MISC

  // This method will only be used at the start of the simulation or after a reset.
  public void addGoodEngineToPool(Engine engineToAdd) {
    goodEnginePool.add(engineToAdd);
  }

  // goodModuleList[] is an array of LinkedLists.  Each LinkedList contains the
  // associated (according to EngineBlueprint) good modules that this ILevel has
  // (modules that are and are not installed in an engine).  For other methods
  // in this class to work correctly, the modules in a specific LinkedList must
  // be in descending timeToHighTime order.  Thus, a module cannot be simply
  // added to the LinkedList.  It must be added and then moved to its correct
  // position in the LinkedList.
  public void addGoodModuleToPool(Module goodModule) {
    if (goodModule == null) {
      throw new IllegalArgumentException ("ILevel Class addGoodModuleToPool Method - " +
        "module to add was null.");
    }
    else                                                                                    if
(goodModuleList[EngineBlueprint.getIndexOfModuleType(goodModule.getType())].contains(goodModul
e)) {
      throw new IllegalArgumentException ("ILevel Class addGoodModuleToPool Method - " +
        "module to add was previously added to the pool.");
    }
    else {
      Module storageModule1 = null;
      Module storageModule2 = null;
      boolean continueLoop = true;
      int moduleIndex = EngineBlueprint.getIndexOfModuleType(goodModule.getType());
      goodModuleList[moduleIndex].add(goodModule);
      for (int index = goodModuleList[moduleIndex].size() - 1; index > 0; index--) {
        if(continueLoop) {
          storageModule1 = (Module)goodModuleList[moduleIndex].get(index);
          storageModule2 = (Module)goodModuleList[moduleIndex].get(index - 1);
          if(storageModule1.getClosestHighTime() > storageModule2.getClosestHighTime()) {
```

```
            goodModuleList[moduleIndex].set(index, storageModule2);
            goodModuleList[moduleIndex].set(index - 1, storageModule1);
          }
          else {
            continueLoop = false;
          }
        }
      }
    }
  }


// Finds the Module with the closest high time greater than timeToMatch.  If no
// Modules have a high time that is greater than timeToMatch, the Module with
// the greatest high time is returned - since the Modules are put into descending
// order in the goodModuleList, this is just the first Module in the associated
// goodModuleList.
private Module findClosestHighTime(double timeToMatch, int moduleIndex) {
  if (timeToMatch < 0) {
    throw new IllegalArgumentException ("ILevel Class findClosestHighTime Method - " +
      "timeToMatch " + timeToMatch + " is less than zero.");
  }
  else if ((moduleIndex < 0) || (moduleIndex >= EngineBlueprint.getNumberOfModules())) {
    throw new IllegalArgumentException ("ILevel Class findClosestHighTime Method - " +
      "moduleIndex " + moduleIndex + " is not within bound.");
  }
  else if (goodModuleList[moduleIndex].size() == 0) {
    throw new IllegalArgumentException ("ILevel Class findClosestHighTime Method - " +
      "Cannot find a closest high time.  No modules in pool " + moduleIndex + ".");
  }
  else {
    Module moduleToReturn = (Module)goodModuleList[moduleIndex].getFirst();
    Module storageModule = null;
    boolean continueLoop = true;
    for (int index=0; index<goodModuleList[moduleIndex].size(); index++) {
      if(continueLoop) {
        storageModule = (Module)goodModuleList[moduleIndex].get(index);
        if (storageModule.getClosestHighTime() > timeToMatch) {
          moduleToReturn = storageModule;
        }
        else {
          continueLoop = false;
        }
      }
    }
    return moduleToReturn;
  }
}

//INSTANCE METHODS - O Level Supported Methods
public void addOLevelSupported(OLevel supportedOLevel) {
  oLevelSupported.add(supportedOLevel);
}

//INSTANCE METHODS - reporting methods
public void reportILevelNISScreenWithReturn() {
```

```
      System.out.print(format0.format(engineNIS) + "    ");
      for (int index = 0; index < numberOfModules; index++) {
        System.out.print(format0.format(moduleNIS[index]) + "    ");
      }
      System.out.println(" ");
    }
  }
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX L. DLEVEL CLASS JAVA CODE

```
/**
 * LCDR Eric J. Schoch
 * F-18 Thesis Work
 * Last Updated: 11 AUG 03
 * <P>
 * Comments: A class to simulate the D-Level in F-18 engine module repair.  This
 * D Level only repairs and returns modules - it has no and does not repair entire
 * engines.
 **/

package jet;
import java.util.*;
import simkit.*;
import simkit.random.*;

public class DLevel extends SimEntityBase {

  //INSTANCE VARIABLES
  public int uic;
  public int[] moduleAllowance;
  public int[] moduleNIS;
  public RandomVariate[] repairRV;
  public LinkedList[] needsModule;
  public int[] numberOfBadModules;
  public LinkedList[] goodModulePool;
  public LinkedList iLevelSupported;

  //CONSTRUCTOR METHODS
  public DLevel(int name, int[] modAll, RandomVariate[] repair) {
    if (modAll.length != EngineBlueprint.getNumberOfModules()) {
      throw new IllegalArgumentException("DLevel Class Constructor Method - " +
        "moduleAllowance does not contain correct number of allowances.");
    }
    else {
      moduleAllowance = new int[EngineBlueprint.getNumberOfModules()];
      moduleNIS = new int[EngineBlueprint.getNumberOfModules()];
    }
    for (int index = 0; index < EngineBlueprint.getNumberOfModules(); index++){
      if (modAll[index] < 0) {
        throw new IllegalArgumentException("DLevel Class Constructor Method - " +
          "moduleAllowance can not be less than zero.");
      }
      else {
        moduleAllowance[index] = modAll[index];
        moduleNIS[index] = 0;
      }
    }
    if (repair.length != EngineBlueprint.getNumberOfModules()) {
      throw new IllegalArgumentException("DLevel Class Constructor Method - " +
        "repairRV does not contain " + EngineBlueprint.getNumberOfModules() +
        " Random Variates.");
    }
```

```
  repairRV = new RandomVariate[EngineBlueprint.getNumberOfModules()];
  needsModule = new LinkedList[EngineBlueprint.getNumberOfModules()];
  numberOfBadModules = new int[EngineBlueprint.getNumberOfModules()];
  goodModulePool = new LinkedList[EngineBlueprint.getNumberOfModules()];
  for (int index = 0; index < EngineBlueprint.getNumberOfModules(); index++) {
    repairRV[index] = repair[index];
    needsModule[index] = new LinkedList();
    numberOfBadModules[index] = 0;
    goodModulePool[index] = new LinkedList();
  }
}

public void reset() {
  super.reset();
  for (int index = 0; index < EngineBlueprint.getNumberOfModules(); index++) {
    moduleNIS[index] = 0;
    needsModule[index].clear();
    numberOfBadModules[index] = 0;
  }
}

public void simulationReset() {
  for (int index = 0; index < EngineBlueprint.getNumberOfModules(); index++) {
    goodModulePool[index].clear();
  }
}


//INSTANCE METHODS
//INSTANCE METHODS - Getter Methods
public int[] getModuleAllowance() { return moduleAllowance; }

public int getSpecificModuleAllowance(int moduleIndex) {
  return moduleAllowance[moduleIndex];
}

//INSTANCE METHODS - to schedule the D Level
public void doILRequestsModuleFromDL(Module moduleNeeded, ILevel whoNeedsIt){
  if (goodModulePool[moduleNeeded.getModuleIndex()].size() > 0) {
    waitDelay("ILRequestedModuleISAtDL", 0.0, new Object[] {moduleNeeded, whoNeedsIt} );
  }
  else {
    waitDelay("ILRequestedModuleNISAtDL", 0.0, new Object[] {moduleNeeded, whoNeedsIt} );
  }
}

public void doILRequestedModuleISAtDL(Module moduleNeeded, ILevel whoNeedsIt){
  waitDelay("StartDLtoILModuleMove", 0.0,
    new Object[] {goodModulePool[moduleNeeded.getModuleIndex()].removeFirst(), whoNeedsIt} );
}

public void doILRequestedModuleNISAtDL(Module moduleNeeded, ILevel whoNeedsIt){
  int moduleIndex = moduleNeeded.getModuleIndex();
  needsModule[moduleIndex].add(whoNeedsIt);
  moduleNIS[moduleIndex]++;
}
```

```
  public void doCompleteILtoDLBadModuleMove(Module badModule){
    numberOfBadModules[badModule.getModuleIndex()]++;
    waitDelay("StartDLModuleRepair", 0.0, badModule);
  }

  public void doStartDLModuleRepair(Module badModule){
    waitDelay("CompleteDLModuleRepair",
      repairRV[badModule.getModuleIndex()].generate(), badModule);
  }

  public void doCompleteDLModuleRepair(Module goodModule){
    goodModule.repairModule();
    int moduleIndex = goodModule.getModuleIndex();
    numberOfBadModules[moduleIndex]--;
    if (needsModule[moduleIndex].size() > 0) {
      waitDelay("DLRepairedModuleNeededAtIL", 0.0, goodModule);
    }
    else {
      waitDelay("DLAddsGoodModuleToPool", 0.0, goodModule);
    }
  }

  public void doDLRepairedModuleNeededAtIL(Module goodModule){
    waitDelay("StartDLtoILModuleMove", 0.0,
      new Object[] {goodModule,
        (ILevel) needsModule[goodModule.getModuleIndex()].removeFirst()} );
  }

  public void doDLAddsGoodModuleToPool(Module goodModule) {
    goodModulePool[goodModule.getModuleIndex()].add(goodModule);
  }

  public void doStartDLtoILModuleMove(Module mod, ILevel whoTo){
  }

  //INSTANCE METHODS - MISC
  public void addGoodModuleToPool(Module goodModule) {
    if (goodModule == null) {
      throw new IllegalArgumentException ("DLevel Class addGoodModuleToPool Method - " +
        "module to add was null.");
    }
    else                                                                    if
(goodModulePool[EngineBlueprint.getIndexOfModuleType(goodModule.getType())].contains(goodModul
e)) {
      throw new IllegalArgumentException ("DLevel Class addGoodModuleToPool Method - " +
        "module to add was previously added to the pool.");
    }
    else {

goodModulePool[EngineBlueprint.getIndexOfModuleType(goodModule.getType())].add(goodModule);
    }
  }

  //INSTANCE METHODS - I Levels Supported
  public void addILevelSupported(ILevel supportedILevel) {
```

```
        iLevelSupported.add(supportedILevel);
    }

}
```

APPENDIX M. F18SIMULATIONMANAGER CLASS JAVA CODE

```
/**
 * LCDR Eric J. Schoch
 * F-18 Thesis Work
 * Last Updated: 15 AUG 03
 * <P>
 * Comments: This class figures out how many modules and engines are needed based
 * on the number of aircraft and number of spares at all I and D levels.  This class
 * "remembers" every object made in the simulation and thus serves as the report
 * generator for the simulation.
 *   This class resets all modules and engines when the simulation is reset (since
 * modules and engines are not SimEntityBases, they have no doReset).  It also
 * clears all parts pools and brings the I and D Levels back up to their
 * authorized allowances.
 **/

package jet;
import java.util.*;
import java.text.DecimalFormat;
import simkit.*;
import simkit.data.SimpleStats2;
import simkit.random.*;

public class F18SimulationManager extends SimEntityBase{

  //INSTANCE VARIABLES
  public int yearsToRun;
  public int yearCounter;
  public RandomVariate[][] moduleFailureRV;
  public LinkedList[] modulePool;
  public LinkedList enginePool;
  public LinkedList aircraftPool;
  public LinkedList oLevelPool;
  public LinkedList iLevelPool;
  public LinkedList dLevelPool;
  public LinkedList[] unusedModules;
  public LinkedList unusedEngines;
  public SimpleStats2[] aoStat;
  public SimpleStats2[] pEngineStat;
  DecimalFormat format0;
  DecimalFormat format1;
  DecimalFormat format2;
  DecimalFormat format3;
  DecimalFormat format4;
  DecimalFormat format5;
  DecimalFormat formatI2;
  DecimalFormat formatD1dot4;

  public F18SimulationManager(int years, RandomVariate[][] modFailRV) {
    yearsToRun = years;
    yearCounter = 0;
    aoStat = new SimpleStats2[yearsToRun];
```

```
    pEngineStat = new SimpleStats2[yearsToRun];
    for (int index = 0; index < yearsToRun; index++) {
      aoStat[index] = new SimpleStats2();
      pEngineStat[index] = new SimpleStats2();
    }
    moduleFailureRV = modFailRV;
    modulePool = new LinkedList[EngineBlueprint.getNumberOfModules()];
    unusedModules = new LinkedList[EngineBlueprint.getNumberOfModules()];
    for (int index = 0; index < EngineBlueprint.getNumberOfModules(); index++) {
      modulePool[index] = new LinkedList();
      unusedModules[index] = new LinkedList();
    }
    enginePool = new LinkedList();
    aircraftPool = new LinkedList();
    oLevelPool = new LinkedList();
    iLevelPool = new LinkedList();
    dLevelPool = new LinkedList();
    unusedEngines = new LinkedList();
    format0 = new DecimalFormat(" 000 ");
    format1 = new DecimalFormat(" 00000.0 ");
    format2 = new DecimalFormat(" 000000000.00 ");
    format3 = new DecimalFormat(" 0.000 ");
    format4 = new DecimalFormat(" 0.0000 ");
    format5 = new DecimalFormat(" 0.00000 ");
    formatI2 = new DecimalFormat(" 00 ");
    formatD1dot4 = new DecimalFormat(" 0.0000 ");
  }

  //INSTANCE METHODS

  //INSTANCE METHODS - to reset the simulation
  public void reset() {
    yearCounter = 0;
    Module storageModule = null;
    for (int index1 = 0; index1 < EngineBlueprint.getNumberOfModules(); index1++){
      unusedModules[index1].clear();
      for (int index2 = 0; index2 < modulePool[index1].size(); index2++){
        storageModule = (Module) modulePool[index1].get(index2);
        storageModule.simulationReset();
        unusedModules[index1].add(storageModule);
      }
    }
    unusedEngines.clear();
    Module[] moduleArray = new Module[EngineBlueprint.getNumberOfModules()];
    for (int index1 = 0; index1 < enginePool.size(); index1++) {
      for (int index2 = 0; index2 < EngineBlueprint.getNumberOfModules(); index2++) {
        moduleArray[index2] = (Module) unusedModules[index2].removeFirst();
      }
      ((Engine)enginePool.get(index1)).simulationReset(moduleArray);
      unusedEngines.add((Engine)enginePool.get(index1));
    }
    //Put working engines into the aircraft.
    for (int index1 = 0; index1 < aircraftPool.size(); index1++) {
      Engine engine1 = (Engine) unusedEngines.removeFirst();
      Engine engine2 = (Engine) unusedEngines.removeFirst();
      ((F18Hornet)aircraftPool.get(index1)).simulationReset(engine1, engine2);
```

```
    }
    //No simulationReset is needed in the OLevel - reset does the job.
    //ILevel
    for (int index1 = 0; index1 < iLevelPool.size(); index1++) {
      ILevel storageILevel = (ILevel)iLevelPool.get(index1);
      storageILevel.simulationReset();
      for (int index2 = 0; index2 < storageILevel.getEngineAllowance(); index2++) {
        storageILevel.addGoodEngineToPool((Engine)unusedEngines.removeFirst());
      }
      for (int index2 = 0; index2 < EngineBlueprint.getNumberOfModules(); index2++) {
        for (int index3 = 0; index3 < storageILevel.getSpecificModuleAllowance(index2);
          index3++) {
            storageILevel.addGoodModuleToPool((Module)unusedModules[index2].removeFirst());
        }
      }
    }
    //DLevel
    for (int index1 = 0; index1 < dLevelPool.size(); index1++) {
      DLevel storageDLevel = (DLevel)dLevelPool.get(index1);
      storageDLevel.simulationReset();
      for (int index2 = 0; index2 < EngineBlueprint.getNumberOfModules(); index2++) {
        for (int index3 = 0; index3 < storageDLevel.getSpecificModuleAllowance(index2);
          index3++) {
            storageDLevel.addGoodModuleToPool((Module)unusedModules[index2].removeFirst());
        }
      }
    }

}

//Instance Methods - to run reports.
public void doRun() {
  waitDelay("AnnualStatUpdate", 24.0*7.0*52.0);
}

public void doAnnualStatUpdate() {
  System.out.println("   Year at for this run : " + (yearCounter+1));
  aoStat[yearCounter].newObservation(this.returnTotalAvailability());
  pEngineStat[yearCounter].newObservation(this.reportILevelEngineNIS0or1());
  yearCounter++;
  waitDelay("AnnualStatUpdate", 24.0*7.0*52.0);
}

//INSTANCE METHODS - to setup and keep tracked of all simulation objects
public DLevel makeDLevel(int name, int[] modAll, RandomVariate[] repair) {
  DLevel newDLevel = new DLevel(name, modAll, repair);
  dLevelPool.add(newDLevel);
  Module allowanceModule = null;
  for (int index1 = 0; index1 < EngineBlueprint.getNumberOfModules(); index1++) {
    for (int index2 = 0; index2 < modAll[index1]; index2++){
      allowanceModule = this.makeModule(index1);
      newDLevel.addGoodModuleToPool(allowanceModule);
    }
  }
  return newDLevel;
}
```

```
public ILevel makeILevel(int name, double xfer, int engAll, int[] modAll) {
  ILevel newILevel = new ILevel(name, xfer, engAll, modAll);
  iLevelPool.add(newILevel);
  Module allowanceModule = null;
  Engine allowanceEngine = null;
  for (int index=0; index < engAll; index++) {
    allowanceEngine = this.makeEngine();
    newILevel.addGoodEngineToPool(allowanceEngine);
  }
  for (int index1 = 0; index1 < EngineBlueprint.getNumberOfModules(); index1++) {
    for (int index2 = 0; index2 < modAll[index1]; index2++){
      allowanceModule = this.makeModule(index1);
      newILevel.addGoodModuleToPool(allowanceModule);
    }
  }
  return newILevel;
}

public OLevel makeOLevel(int name, boolean can, double et) {
  OLevel newOLevel = new OLevel(name, can, et);
  oLevelPool.add(newOLevel);
  return newOLevel;
}

public F18Hornet makeF18Hornet(double[] flightSchedule, int delay) {
  // To output to screen each F18's flight schedule when created,
  // remove the comment marks from the next four lines.
  //for (int index = 0; index < flightSchedule.length; index ++) {
  //   System.out.print(flightSchedule[index] + " " );
  //}
  //System.out.println(" ");
  Engine engine1 = this.makeEngine();
  Engine engine2 = this.makeEngine();
  F18Hornet newF18Hornet = new F18Hornet(aircraftPool.size() + 1, engine1, engine2,
    flightSchedule, delay);
  aircraftPool.add(newF18Hornet);
  return newF18Hornet;
}

public Engine makeEngine() {
  Module[] modulesToPass = new Module[EngineBlueprint.getNumberOfModules()];
  for (int index = 0; index < EngineBlueprint.getNumberOfModules(); index++) {
    modulesToPass[index] = this.makeModule(index);
  }
  Engine newEngine = new Engine(enginePool.size() + 1, modulesToPass);
  enginePool.add(newEngine);
  return newEngine;
}

public Module makeModule(int moduleIndex) {
  Module newModule = new Module(EngineBlueprint.getModuleTypeOfIndex(moduleIndex),
    modulePool[moduleIndex].size() + 1, moduleFailureRV[moduleIndex]);
  modulePool[moduleIndex].add(newModule);
  return newModule;
}
```

```
  //INSTANCE METHODS - reports
  public void reportAoAndPEngByYear() {
    System.out.println(" ");
    System.out.println("Years to run:  " + yearsToRun);
    System.out.println("Runs per year: " + aoStat[0].getCount());
    System.out.println(" ");
    System.out.println("Years Ran  Ao Mean   Ao Min    Ao Max    Ao Var    Ao StdDv  P(Eng)");
    System.out.println("=========    ========    ========    ========    ========    ========
=========");
    for (int index = 0; index < yearsToRun; index++){
      System.out.println( "   " + formatI2.format(index + 1) + "    " +
        formatD1dot4.format(aoStat[index].getMean()) + "  " +
        formatD1dot4.format(aoStat[index].getMinObs()) + "  " +
        formatD1dot4.format(aoStat[index].getMaxObs()) + "  " +
        formatD1dot4.format(aoStat[index].getVariance()) + "  " +
        formatD1dot4.format(aoStat[index].getStandardDeviation()) + "   " +
        formatD1dot4.format(pEngineStat[index].getMean()) );
    }
  }

  public void reportIndividualAircraftAvailability(){

    F18Hornet storageF18 = null;
    double totalMsnHrsMsd = 0;     // Total mission hours missed
    double totalMsnHrsFln = 0;     // Total mission hours flown
    double totalTimeUnavl = 0;     // Total time unavailable
    double totalTimeAvail = 0;     // Total time available
    double totalTotalTime = 0;     // Total total time in simulation
    double storageArray[] = null;

    System.out.println("Serno  Msn Hrs Missd  Msn Hrs Flown  Time UnAvail  Time Availbl    Total Time
Avail(Ao)");
    System.out.println("-----  -------------  -------------  ------------  ------------   ------------   ---------");
    for (int index = 0; index < aircraftPool.size(); index++){
      storageF18 = (F18Hornet) aircraftPool.get(index);
      storageArray = storageF18.reportAircraftAvailabilityScreenWithReturn();
      totalMsnHrsMsd += storageArray[0];
      totalMsnHrsFln += storageArray[1];
      totalTimeUnavl += storageArray[2];
      totalTimeAvail += storageArray[3];
      totalTotalTime += storageArray[4];
    }
    System.out.println("-----  -------------  -------------  ------------  ------------   ------------   ---------");
    System.out.println( "TOTAL  " +
        format2.format(totalMsnHrsMsd) + " " +
        format2.format(totalMsnHrsFln) +
        format2.format(totalTimeUnavl) +
        format2.format(totalTimeAvail) + " " +
        format2.format(totalTotalTime) + "   " +
        format4.format(totalTimeAvail/totalTotalTime));
  }

  public void reportTotalAvailability() {
    F18Hornet storageF18 = null;
    double totalMsnHrsMsd = 0;     // Total mission hours missed
```

```
  double totalMsnHrsFln = 0;     // Total mission hours flown
  double totalTimeUnavl = 0;     // Total time unavailable
  double totalTimeAvail = 0;     // Total time available
  double totalTotalTime = 0;     // Total total time in simulation
  double storageArray[] = null;
  for (int index = 0; index < aircraftPool.size(); index++){
    storageF18 = (F18Hornet) aircraftPool.get(index);
    storageArray = storageF18.reportAircraftAvailabilityReturnOnly();
    totalMsnHrsMsd += storageArray[0];
    totalMsnHrsFln += storageArray[1];
    totalTimeUnavl += storageArray[2];
    totalTimeAvail += storageArray[3];
    totalTotalTime += storageArray[4];
  }
  System.out.println( "TOTAL  " +
       format2.format(totalMsnHrsMsd) + " " +
       format2.format(totalMsnHrsFln) +
       format2.format(totalTimeUnavl) +
       format2.format(totalTimeAvail) + " " +
       format2.format(totalTotalTime) + "    " +
       format4.format(totalTimeAvail/totalTotalTime));
}

public double returnTotalAvailability() {
  F18Hornet storageF18 = null;
  double totalMsnHrsMsd = 0;     // Total mission hours missed
  double totalMsnHrsFln = 0;     // Total mission hours flown
  double totalTimeUnavl = 0;     // Total time unavailable
  double totalTimeAvail = 0;     // Total time available
  double totalTotalTime = 0;     // Total total time in simulation
  double storageArray[] = null;
  for (int index = 0; index < aircraftPool.size(); index++){
    storageF18 = (F18Hornet) aircraftPool.get(index);
    storageArray = storageF18.reportAircraftAvailabilityReturnOnly();
    totalMsnHrsMsd += storageArray[0];
    totalMsnHrsFln += storageArray[1];
    totalTimeUnavl += storageArray[2];
    totalTimeAvail += storageArray[3];
    totalTotalTime += storageArray[4];
  }
  return (totalTimeAvail/totalTotalTime);
}

public double reportILevelEngineNIS0or1() {
  int returnValue = 1;
  ILevel storageILevel = null;
  for (int index = 0; index < iLevelPool.size(); index++) {
    storageILevel = (ILevel) iLevelPool.get(index);
    if (storageILevel.getEngineNIS() > 0) {
      returnValue = 0;
    }
  }
  return (double) returnValue;
}
```

```
  public void reportILevelNIS() {
    ILevel storageILevel = null;
    for (int index = 0; index < iLevelPool.size(); index++) {
      storageILevel = (ILevel) iLevelPool.get(index);
      storageILevel.reportILevelNISScreenWithReturn();
    }
  }


}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX N. F18SIMULATIONRANDOMNESS CLASS JAVA CODE

```
/**
 * LCDR Eric J. Schoch
 * F-18 Thesis Work
 * Last Updated: 16 AUG 03
 * <P>
 * Comment:  Any randomness simulated is developed in this class.
 */

package jet;
import java.io.*;
import java.util.*;
import simkit.random.*;

public class F18SimulationRandomness {

  //INSTANCE VARIABLES
  //INSTANCE VARIABLES - general use variables
  int numberOfModules;
  RandomNumber rand;
  RandomVariate[] storageRV;
  RandomVariate uniformRV;

  //INSTRANCE VARIABLES - depot level repair time arrays
  double[][] depotAWMTimes;
  double[][] depotAWPTimes;
  double[][] depotInworkTimes;
  double[][] depotOtherTimes;

  //INSTANCE VARIABLES - depot level repair time RandomVariates
  RandomVariate[] depotAWMRV;
  RandomVariate[] depotAWPRV;
  RandomVariate[] depotInworkRV;
  RandomVariate[] depotOtherRV;
  RandomVariate[] depotRepairRV;

  //INSTANCE VARIABLE - module failure time RandomVariates
  RandomVariate[][] moduleFailureRV;

  //CONSTRUCTOR METHODS
  public F18SimulationRandomness() {

    RandomVariateFactory.addSearchPackage("jet");

    //general use variables
    numberOfModules = EngineBlueprint.getNumberOfModules();
    rand = RandomNumberFactory.getInstance(CongruentialSeeds.SEED [4]);
    storageRV = new RandomVariate[4];
    uniformRV = RandomVariateFactory.getInstance("Uniform",
        new Object[] {new Double(0), new Double(1.0)}, rand);
```

```
//depot level repair time arrays
depotAWMTimes = new double[numberOfModules][];
depotAWPTimes = new double[numberOfModules][];
depotInworkTimes = new double[numberOfModules][];
depotOtherTimes = new double[numberOfModules][];

//depot level repair time RandomVariates
depotAWMRV = new RandomVariate[numberOfModules];
depotAWPRV = new RandomVariate[numberOfModules];
depotInworkRV = new RandomVariate[numberOfModules];
depotOtherRV = new RandomVariate[numberOfModules];
depotRepairRV = new RandomVariate[numberOfModules];

//read in the depot repair times
//Depot reapir times are broken down into four tracked categories:
// AWM = awaiting maintenance
// AWP = awaiting parts
// Inwork = the time the module is actually undergoing repair
// Other = a catch all for investigations and misc delay times
depotAWMTimes[0] = this.getDataArray("H:/F18DataFiles/Depot0FanAWM.txt");
depotAWPTimes[0] = this.getDataArray("H:/F18DataFiles/Depot0FanAWP.txt");
depotInworkTimes[0] = this.getDataArray("H:/F18DataFiles/Depot0FanInwork.txt");
depotOtherTimes[0] = this.getDataArray("H:/F18DataFiles/Depot0FanOther.txt");
depotAWMTimes[1] = this.getDataArray("H:/F18DataFiles/Depot1CompressorAWM.txt");
depotAWPTimes[1] = this.getDataArray("H:/F18DataFiles/Depot1CompressorAWP.txt");
depotInworkTimes[1] = this.getDataArray("H:/F18DataFiles/Depot1CompressorInwork.txt");
depotOtherTimes[1] = this.getDataArray("H:/F18DataFiles/Depot1CompressorOther.txt");
depotAWMTimes[2] = this.getDataArray("H:/F18DataFiles/Depot2CombustorAWM.txt");
depotAWPTimes[2] = this.getDataArray("H:/F18DataFiles/Depot2CombustorAWP.txt");
depotInworkTimes[2] = this.getDataArray("H:/F18DataFiles/Depot2CombustorInwork.txt");
depotOtherTimes[2] = this.getDataArray("H:/F18DataFiles/Depot2CombustorOther.txt");
depotAWMTimes[3] = this.getDataArray("H:/F18DataFiles/Depot3HPTAWM.txt");
depotAWPTimes[3] = this.getDataArray("H:/F18DataFiles/Depot3HPTAWP.txt");
depotInworkTimes[3] = this.getDataArray("H:/F18DataFiles/Depot3HPTInwork.txt");
depotOtherTimes[3] = this.getDataArray("H:/F18DataFiles/Depot3HPTOther.txt");
depotAWMTimes[4] = this.getDataArray("H:/F18DataFiles/Depot4LPTAWM.txt");
depotAWPTimes[4] = this.getDataArray("H:/F18DataFiles/Depot4LPTAWP.txt");
depotInworkTimes[4] = this.getDataArray("H:/F18DataFiles/Depot4LPTInwork.txt");
depotOtherTimes[4] = this.getDataArray("H:/F18DataFiles/Depot4LPTOther.txt");
depotAWMTimes[5] = this.getDataArray("H:/F18DataFiles/Depot5AfterburnerAWM.txt");
depotAWPTimes[5] = this.getDataArray("H:/F18DataFiles/Depot5AfterburnerAWP.txt");
depotInworkTimes[5] = this.getDataArray("H:/F18DataFiles/Depot5AfterburnerInwork.txt");
depotOtherTimes[5] = this.getDataArray("H:/F18DataFiles/Depot5AfterburnerOther.txt");

//takes the arrays read in above and makes RandomVariates out of them
//Note: this simulation uses actual failure times and draws from those times
//      with replacement
for (int index = 0; index < numberOfModules; index++) {
  depotAWMRV[index] = RandomVariateFactory.getInstance("Resample",
    new Object[] {depotAWMTimes[index]});
  storageRV[0] = depotAWMRV[index];
  depotAWPRV[index] = RandomVariateFactory.getInstance("Resample",
    new Object[] {depotAWPTimes[index]});
  storageRV[1] = depotAWPRV[index];
  depotInworkRV[index] = RandomVariateFactory.getInstance("Resample",
    new Object[] {depotInworkTimes[index]});
```

```
    storageRV[2] = depotInworkRV[index];
    depotRepairRV[index] = RandomVariateFactory.getInstance("Resample",
      new Object[] {depotOtherTimes[index]});
    storageRV[3] = depotRepairRV[index];
    depotRepairRV[index] = RandomVariateFactory.getInstance("Convolution",
      new Object[]{storageRV}, rand );
  }

  //module failure time RandomVariates
  moduleFailureRV = new RandomVariate[numberOfModules][];

  for (int index = 0; index < numberOfModules; index++){
    moduleFailureRV[index] = new RandomVariate[2];
  }

  moduleFailureRV[0][0] = RandomVariateFactory.getInstance("Resample",
    new Object[] {this.getDataArray("H:/F18DataFiles/Module0FanFailure1.txt")},
    rand);
  moduleFailureRV[0][1] = RandomVariateFactory.getInstance("Resample",
    new Object[] {this.getDataArray("H:/F18DataFiles/Module0FanFailure2.txt")},
    rand);
  moduleFailureRV[1][0] = RandomVariateFactory.getInstance("Resample",
    new Object[] {this.getDataArray("H:/F18DataFiles/Module1CompressorFailure1.txt")},
    rand);
  moduleFailureRV[1][1] = RandomVariateFactory.getInstance("Resample",
    new Object[] {this.getDataArray("H:/F18DataFiles/Module1CompressorFailure2.txt")},
    rand);
  moduleFailureRV[2][0] = RandomVariateFactory.getInstance("Resample",
    new Object[] {this.getDataArray("H:/F18DataFiles/Module2CombustorFailure1.txt")},
    rand);
  moduleFailureRV[2][1] = RandomVariateFactory.getInstance("Resample",
    new Object[] {this.getDataArray("H:/F18DataFiles/Module2CombustorFailure2.txt")},
    rand);
  moduleFailureRV[3][0] = RandomVariateFactory.getInstance("Resample",
    new Object[] {this.getDataArray("H:/F18DataFiles/Module3HPTFailure1.txt")},
    rand);
  moduleFailureRV[3][1] = RandomVariateFactory.getInstance("Resample",
    new Object[] {this.getDataArray("H:/F18DataFiles/Module3HPTFailure2.txt")},
    rand);
  moduleFailureRV[4][0] = RandomVariateFactory.getInstance("Resample",
    new Object[] {this.getDataArray("H:/F18DataFiles/Module4LPTFailure1.txt")},
    rand);
  moduleFailureRV[4][1] = RandomVariateFactory.getInstance("Resample",
    new Object[] {this.getDataArray("H:/F18DataFiles/Module4LPTFailure2.txt")},
    rand);
  moduleFailureRV[5][0] = RandomVariateFactory.getInstance("Resample",
    new Object[] {this.getDataArray("H:/F18DataFiles/Module5AfterburnerFailure1.txt")},
    rand);
  moduleFailureRV[5][1] = RandomVariateFactory.getInstance("Resample",
    new Object[] {this.getDataArray("H:/F18DataFiles/Module5AfterburnerFailure2.txt")},
    rand);
}

//INSTANCE METHODS
//INSTANCE METHODS - getter methods
public RandomVariate getUniformRV() { return uniformRV; }
```

```
public RandomVariate[] getAllDepotRepairRV() { return depotRepairRV; }

public RandomVariate getSpecificDepotRepairRV(int moduleIndex) {
  if ((moduleIndex >= numberOfModules) || (moduleIndex < 0)) {
    throw new IllegalArgumentException("F18SimulationRandomness Class " +
        "getDepotRepairRV Method - moduleIndex passed is out of bounds");
  }
  else{
    return depotRepairRV[moduleIndex];
  }
}

public RandomVariate[][] getAllModuleFailureRV(){ return moduleFailureRV; }

public RandomVariate[] getSpecificModuleFailureRV(int moduleIndex) {
  if ((moduleIndex >= numberOfModules) || (moduleIndex < 0)) {
    throw new IllegalArgumentException("F18SimulationRandomness Class " +
        "getModuleFailureRV Method - moduleIndex passed is out of bounds");
  }
  else{
    return moduleFailureRV[moduleIndex];
  }
}

//INSTANCE METHODS - used to ready in data files and convert them to arrays
public double[] getDataArray(String filename) {
  double[] storageArray;
  try {
    storageArray = this.getData(filename);
  }
  catch (IOException e) {
    throw new RuntimeException("F18SimulationRandomness Class getDataArray Method - " +
      filename + " could not be found.");
  }
  return storageArray;
}

public double[] getData(String filename) throws IOException {
  BufferedReader in = new BufferedReader( new InputStreamReader(
    new FileInputStream(filename)));
  return readFile(in);
}

public double[] readFile(BufferedReader input) throws IOException{
  String aLine;
  StringTokenizer stringTokens;
  LinkedList data = new LinkedList();
  double[] arrayToReturn;
  while ((aLine = input.readLine()) != null) {
    if((aLine.length() > 0)  && (aLine.charAt(0) != 'c')) {
      stringTokens = new StringTokenizer(aLine);
      data.add(stringTokens.nextToken());
    }
  }
  input.close();
```

```
    arrayToReturn = new double[data.size()];
    for (int index = 0; index < arrayToReturn.length; index++) {
      arrayToReturn[index] = Double.parseDouble((String)data.get(index));
    }
    return arrayToReturn;
  }

}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX O. F18SIMULATIONSETUP CLASS JAVA CODE

```
/**
 * LCDR Eric J. Schoch
 * F-18 Thesis Work
 * Last Updated: 08 AUG 03
 * <P>
 * Comment:  This controls the entire simulation.
 */

package jet;
import java.util.*;
import simkit.*;
import simkit.random.*;

public class F18SimulationSetup {


  public static void main(String args[]) {

    int totalYearsToRun = 25;
    int totalReplicationsPerRun = 100;

    int numberOfModules = EngineBlueprint.getNumberOfModules();
    F18SimulationRandomness simRandom = new F18SimulationRandomness();
    F18SimulationManager simManager = new F18SimulationManager(totalYearsToRun,
simRandom.getAllModuleFailureRV());
    FlightSchedules f18Schedule = new FlightSchedules(simRandom.getUniformRV());

    boolean cannibalize = true;

    //Storage Variables used to temporarily store  an object.
    DLevel activeDLevel = null;
    ILevel activeILevel = null;
    OLevel activeOLevel = null;
    F18Hornet activeF18 = null;

    //Make NADEP Jacksonville (D Level)
    activeDLevel = simManager.makeDLevel(99999,new int[] {0,0,0,0,0,0},
      simRandom.getAllDepotRepairRV());

    //Make PAX River (I Level)
    activeILevel = simManager.makeILevel(10000, 120.0, 8, new int[] {7,3,4,2,1,2});
    activeDLevel.addSimEventListener(activeILevel);
    activeILevel.addSimEventListener(activeDLevel);
    //Make a single O Level at PAX River
    activeOLevel = simManager.makeOLevel(10001, cannibalize, 0.0);
    activeILevel.addSimEventListener(activeOLevel);
    activeOLevel.addSimEventListener(activeILevel);
    //Make the F18's that are supported by PAX River
    for (int index = 0; index < 25; index++) {
      activeF18 = simManager.makeF18Hornet(f18Schedule.getRandomThreeDayBlock(2.75),index*0);
      activeOLevel.addSimEventListener(activeF18);
      activeF18.addSimEventListener(activeOLevel);
```

```
}
//Make NAS LEMOORE (I Level)
activeILevel = simManager.makeILevel(20000, 120.0, 7, new int[] {8,4,5,3,1,2});
activeDLevel.addSimEventListener(activeILevel);
activeILevel.addSimEventListener(activeDLevel);
//Make a single O Level
activeOLevel = simManager.makeOLevel(20001, cannibalize, 0.0);
activeILevel.addSimEventListener(activeOLevel);
activeOLevel.addSimEventListener(activeILevel);
//Make the F18's that are supported by PAX River
for (int index = 0; index < 30; index++) {
  activeF18 = simManager.makeF18Hornet(f18Schedule.getRandomThreeDayBlock(2.75),index*0);
  activeOLevel.addSimEventListener(activeF18);
  activeF18.addSimEventListener(activeOLevel);
}
//Make NAS OCEANA (I Level)
activeILevel = simManager.makeILevel(30000, 120.0, 5, new int[] {6,3,4,2,1,2});
activeDLevel.addSimEventListener(activeILevel);
activeILevel.addSimEventListener(activeDLevel);
//Make a single O Level
activeOLevel = simManager.makeOLevel(30001, cannibalize, 0.0);
activeILevel.addSimEventListener(activeOLevel);
activeOLevel.addSimEventListener(activeILevel);
//Make the F18's that are supported by PAX River
for (int index = 0; index < 22; index++) {
  activeF18 = simManager.makeF18Hornet(f18Schedule.getRandomThreeDayBlock(2.75),index*0);
  activeOLevel.addSimEventListener(activeF18);
  activeF18.addSimEventListener(activeOLevel);
}
//Make CV01 (I Level)
activeILevel = simManager.makeILevel(40000, 360.0, 8, new int[] {9,5,6,4,1,2});
activeDLevel.addSimEventListener(activeILevel);
activeILevel.addSimEventListener(activeDLevel);
//Make a single O Level
activeOLevel = simManager.makeOLevel(40001, cannibalize, 0.0);
activeILevel.addSimEventListener(activeOLevel);
activeOLevel.addSimEventListener(activeILevel);
//Make the F18's that are supported by PAX River
for (int index = 0; index < 26; index++) {
  activeF18 = simManager.makeF18Hornet(f18Schedule.getRandomThreeDayBlock(2.75),index*0);
  activeOLevel.addSimEventListener(activeF18);
  activeF18.addSimEventListener(activeOLevel);
}
//Make CV02 (I Level)
activeILevel = simManager.makeILevel(50000, 360.0, 8, new int[] {9,5,6,4,1,2});
activeDLevel.addSimEventListener(activeILevel);
activeILevel.addSimEventListener(activeDLevel);
//Make a single O Level
activeOLevel = simManager.makeOLevel(50001, cannibalize, 0.0);
activeILevel.addSimEventListener(activeOLevel);
activeOLevel.addSimEventListener(activeILevel);
//Make the F18's that are supported by PAX River
for (int index = 0; index < 26; index++) {
  activeF18 = simManager.makeF18Hornet(f18Schedule.getRandomThreeDayBlock(2.75),index*0);
  activeOLevel.addSimEventListener(activeF18);
  activeF18.addSimEventListener(activeOLevel);
```

```
}
//Make CV03 (I Level)
activeILevel = simManager.makeILevel(60000, 360.0, 4, new int[] {5,2,4,2,0,1});
activeDLevel.addSimEventListener(activeILevel);
activeILevel.addSimEventListener(activeDLevel);
//Make a single O Level
activeOLevel = simManager.makeOLevel(60001, cannibalize, 0.0);
activeILevel.addSimEventListener(activeOLevel);
activeOLevel.addSimEventListener(activeILevel);
//Make the F18's that are supported by PAX River
for (int index = 0; index < 12; index++) {
  activeF18 = simManager.makeF18Hornet(f18Schedule.getRandomThreeDayBlock(2.75),index*0);
  activeOLevel.addSimEventListener(activeF18);
  activeF18.addSimEventListener(activeOLevel);
}
//Make CV04 (I Level)
activeILevel = simManager.makeILevel(70000, 360.0, 8, new int[] {9,5,6,4,1,2});
activeDLevel.addSimEventListener(activeILevel);
activeILevel.addSimEventListener(activeDLevel);
//Make a single O Level
activeOLevel = simManager.makeOLevel(70001, cannibalize, 0.0);
activeILevel.addSimEventListener(activeOLevel);
activeOLevel.addSimEventListener(activeILevel);
//Make the F18's that are supported by PAX River
for (int index = 0; index < 26; index++) {
  activeF18 = simManager.makeF18Hornet(f18Schedule.getRandomThreeDayBlock(2.75),index*0);
  activeOLevel.addSimEventListener(activeF18);
  activeF18.addSimEventListener(activeOLevel);
}
//Make CV05 (I Level)
activeILevel = simManager.makeILevel(80000, 360.0, 8, new int[] {9,5,6,4,1,2});
activeDLevel.addSimEventListener(activeILevel);
activeILevel.addSimEventListener(activeDLevel);
//Make a single O Level
activeOLevel = simManager.makeOLevel(80001, cannibalize, 0.0);
activeILevel.addSimEventListener(activeOLevel);
activeOLevel.addSimEventListener(activeILevel);
//Make the F18's that are supported by PAX River
for (int index = 0; index < 26; index++) {
  activeF18 = simManager.makeF18Hornet(f18Schedule.getRandomThreeDayBlock(2.75),index*0);
  activeOLevel.addSimEventListener(activeF18);
  activeF18.addSimEventListener(activeOLevel);
}
//Make CV06 (I Level)
activeILevel = simManager.makeILevel(90000, 360.0, 4, new int[] {5,2,4,2,0,1});
activeDLevel.addSimEventListener(activeILevel);
activeILevel.addSimEventListener(activeDLevel);
//Make a single O Level
activeOLevel = simManager.makeOLevel(90001, cannibalize, 0.0);
activeILevel.addSimEventListener(activeOLevel);
activeOLevel.addSimEventListener(activeILevel);
//Make the F18's that are supported by PAX River
for (int index = 0; index < 12; index++) {
  activeF18 = simManager.makeF18Hornet(f18Schedule.getRandomThreeDayBlock(2.75),index*0);
  activeOLevel.addSimEventListener(activeF18);
  activeF18.addSimEventListener(activeOLevel);
```

```
    }
    //Make CV07 (I Level)
    activeILevel = simManager.makeILevel(10900, 360.0, 4, new int[] {5,2,4,2,0,1});
    activeDLevel.addSimEventListener(activeILevel);
    activeILevel.addSimEventListener(activeDLevel);
    //Make a single O Level
    activeOLevel = simManager.makeOLevel(10901, cannibalize, 0.0);
    activeILevel.addSimEventListener(activeOLevel);
    activeOLevel.addSimEventListener(activeILevel);
    //Make the F18's that are supported by PAX River
    for (int index = 0; index < 12; index++) {
      activeF18 = simManager.makeF18Hornet(f18Schedule.getRandomThreeDayBlock(2.75),index*0);
      activeOLevel.addSimEventListener(activeF18);
      activeF18.addSimEventListener(activeOLevel);
    }

    System.out.println("Length of Run (Years) : " + totalYearsToRun);
    for (int index = 0; index < totalReplicationsPerRun; index++){
      System.out.println("Run # : " + (index+1) + " of " + totalReplicationsPerRun);
      Schedule.setVerbose(false);
      Schedule.stopAtTime(24.0*7.0*52.0*totalYearsToRun + 0.01);
      Schedule.reset();
      Schedule.setSingleStep(false);
      Schedule.startSimulation();
    }
    simManager.reportAoAndPEngByYear();
  }

}
```

APPENDIX P. BASELINE1 SIMULATION OUTPUT

Run Name:	Baseline1					
Changes Made:	None					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9909	0.9805	0.9967	0.0000	0.0032	0.0000
2	0.9235	0.9096	0.9469	0.0001	0.0076	0.0000
3	0.8463	0.8312	0.8638	0.0000	0.0062	0.0000
4	0.7832	0.7717	0.8011	0.0000	0.0059	0.0000
5	0.7603	0.7491	0.7752	0.0000	0.0054	0.0000
6	0.7426	0.7332	0.7524	0.0000	0.0046	0.0000
7	0.7214	0.7108	0.7292	0.0000	0.0040	0.0000
8	0.6986	0.6899	0.7066	0.0000	0.0036	0.0000
9	0.6824	0.6729	0.6917	0.0000	0.0037	0.0000
10	0.6720	0.6634	0.6807	0.0000	0.0035	0.0000
11	0.6662	0.6575	0.6752	0.0000	0.0036	0.0000
12	0.6638	0.6540	0.6720	0.0000	0.0034	0.0000
13	0.6613	0.6525	0.6690	0.0000	0.0032	0.0000
14	0.6573	0.6500	0.6644	0.0000	0.0029	0.0000
15	0.6525	0.6457	0.6582	0.0000	0.0028	0.0000
16	0.6475	0.6403	0.6531	0.0000	0.0026	0.0000
17	0.6431	0.6355	0.6482	0.0000	0.0025	0.0000
18	0.6396	0.6319	0.6447	0.0000	0.0025	0.0000
19	0.6371	0.6307	0.6417	0.0000	0.0025	0.0000
20	0.6354	0.6287	0.6395	0.0000	0.0024	0.0000
21	0.6342	0.6281	0.6387	0.0000	0.0024	0.0000
22	0.6332	0.6265	0.6374	0.0000	0.0023	0.0000
23	0.6321	0.6265	0.6356	0.0000	0.0022	0.0000
24	0.6309	0.6242	0.6352	0.0000	0.0022	0.0000
25	0.6297	0.6231	0.6343	0.0000	0.0022	0.0000

Table 14 Baseline1 Simulation Output

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX Q. EXPERIMENT 1 (STAGGERED ENTRY) OUTPUT

Year	0 Weeks	1 Week	Baseline1	3 Weeks	4 Weeks	5 Weeks
1	0.9647	0.9820	0.9909	0.9936	0.9954	0.9964
2	0.8919	0.9053	0.9235	0.9418	0.9561	0.9677
3	0.8042	0.8221	0.8463	0.8691	0.8866	0.9059
4	0.7725	0.7768	0.7832	0.7945	0.8116	0.8339
5	0.7517	0.7552	0.7603	0.7681	0.7766	0.7900
6	0.7338	0.7372	0.7426	0.7493	0.7566	0.7664
7	0.7099	0.7142	0.7214	0.7291	0.7369	0.7465
8	0.6904	0.6931	0.6986	0.7046	0.7122	0.7219
9	0.6767	0.6784	0.6824	0.6874	0.6933	0.7013
10	0.6685	0.6691	0.6720	0.6756	0.6802	0.6868
11	0.6644	0.6642	0.6662	0.6687	0.6723	0.6775
12	0.6627	0.6624	0.6638	0.6658	0.6684	0.6729
13	0.6595	0.6598	0.6613	0.6634	0.6661	0.6700
14	0.6553	0.6561	0.6573	0.6598	0.6625	0.6633
15	0.6502	0.6511	0.6525	0.6552	0.6578	0.6618
16	0.6452	0.6463	0.6475	0.6499	0.6526	0.6563
17	0.6412	0.6420	0.6431	0.6453	0.6478	0.6512
18	0.6381	0.6387	0.6396	0.6414	0.6436	0.6468
19	0.6360	0.6363	0.6371	0.6386	0.6405	0.6434
20	0.6346	0.6348	0.6354	0.6368	0.6384	0.6410
21	0.6335	0.6337	0.6342	0.6355	0.6370	0.6394
22	0.6324	0.6326	0.6332	0.6343	0.6358	0.6381
23	0.6312	0.6313	0.6321	0.6330	0.6346	0.6367
24	0.6300	0.6301	0.6309	0.6319	0.6333	0.6353
25	0.6289	0.6291	0.6297	0.6307	0.6320	0.6340

Table 15 Baseline1 Change - Week Aircraft Entry Stagger Summary of Mean A_0

Run Name:	Baseline1					
Changes Made:	0 Week Stagger between aircraft entry at O-Level					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9647	0.9488	0.9785	0.0000	0.0061	0.0000
2	0.8919	0.8673	0.9136	0.0001	0.0087	0.0000
3	0.8042	0.7831	0.8245	0.0000	0.0067	0.0000
4	0.7725	0.7528	0.7878	0.0000	0.0055	0.0000
5	0.7517	0.7329	0.7649	0.0000	0.0047	0.0000
6	0.7338	0.7163	0.7451	0.0000	0.0044	0.0000
7	0.7099	0.6933	0.7187	0.0000	0.0041	0.0000
8	0.6904	0.6769	0.7008	0.0000	0.0036	0.0000
9	0.6767	0.6631	0.6851	0.0000	0.0036	0.0000
10	0.6685	0.6569	0.6778	0.0000	0.0034	0.0000
11	0.6644	0.6536	0.6741	0.0000	0.0034	0.0000
12	0.6627	0.6527	0.6720	0.0000	0.0031	0.0000
13	0.6595	0.6498	0.6675	0.0000	0.0030	0.0000
14	0.6553	0.6464	0.6622	0.0000	0.0028	0.0000
15	0.6502	0.6424	0.6561	0.0000	0.0027	0.0000
16	0.6452	0.6382	0.6512	0.0000	0.0026	0.0000
17	0.6412	0.6335	0.6463	0.0000	0.0026	0.0000
18	0.6381	0.6302	0.6432	0.0000	0.0025	0.0000
19	0.6360	0.6287	0.6409	0.0000	0.0024	0.0000
20	0.6346	0.6269	0.6397	0.0000	0.0024	0.0000
21	0.6335	0.6264	0.6388	0.0000	0.0024	0.0000
22	0.6324	0.6252	0.6374	0.0000	0.0023	0.0000
23	0.6312	0.6235	0.6360	0.0000	0.0023	0.0000
24	0.6300	0.6226	0.6346	0.0000	0.0023	0.0000
25	0.6289	0.6222	0.6334	0.0000	0.0023	0.0000

Table 16 Baseline1 Change - 0 Week Aircraft Entry Stagger

Run Name:	Baseline1					
Changes Made:	1 Week Stagger between aircraft entry at O-Level					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9820	0.9661	0.9918	0.0000	0.0044	0.0000
2	0.9053	0.8769	0.9208	0.0001	0.0085	0.0000
3	0.8221	0.8027	0.8346	0.0000	0.0059	0.0000
4	0.7768	0.7622	0.7901	0.0000	0.0059	0.0000
5	0.7552	0.7448	0.7656	0.0000	0.0051	0.0000
6	0.7372	0.7269	0.7501	0.0000	0.0047	0.0000
7	0.7142	0.7050	0.7261	0.0000	0.0039	0.0000
8	0.6931	0.6844	0.7062	0.0000	0.0042	0.0000
9	0.6784	0.6701	0.6898	0.0000	0.0038	0.0000
10	0.6691	0.6609	0.6801	0.0000	0.0038	0.0000
11	0.6642	0.6558	0.6743	0.0000	0.0037	0.0000
12	0.6624	0.6534	0.6721	0.0000	0.0037	0.0000
13	0.6598	0.6513	0.6685	0.0000	0.0035	0.0000
14	0.6561	0.6484	0.6633	0.0000	0.0032	0.0000
15	0.6511	0.6440	0.6578	0.0000	0.0029	0.0000
16	0.6463	0.6397	0.6527	0.0000	0.0028	0.0000
17	0.6420	0.6331	0.6481	0.0000	0.0028	0.0000
18	0.6387	0.6316	0.6445	0.0000	0.0027	0.0000
19	0.6363	0.6295	0.6411	0.0000	0.0026	0.0000
20	0.6348	0.6287	0.6398	0.0000	0.0026	0.0000
21	0.6337	0.6287	0.6387	0.0000	0.0024	0.0000
22	0.6326	0.6275	0.6375	0.0000	0.0024	0.0000
23	0.6313	0.6263	0.6365	0.0000	0.0024	0.0000
24	0.6301	0.6255	0.6354	0.0000	0.0023	0.0000
25	0.6291	0.6239	0.6345	0.0000	0.0023	0.0000

Table 17 Baseline1 Change - 1 Week Aircraft Entry Stagger

Run Name:	Baseline1					
Changes Made:	3 Week Stagger between aircraft entry at O-Level					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9936	0.9844	0.9973	0.0000	0.0025	0.0000
2	0.9418	0.9255	0.9637	0.0001	0.0081	0.0000
3	0.8691	0.8540	0.8884	0.0000	0.0068	0.0000
4	0.7945	0.7818	0.8104	0.0000	0.0064	0.0000
5	0.7681	0.7561	0.7831	0.0000	0.0056	0.0000
6	0.7493	0.7373	0.7629	0.0000	0.0056	0.0000
7	0.7291	0.7159	0.7417	0.0000	0.0050	0.0000
8	0.7046	0.6925	0.7158	0.0000	0.0042	0.0000
9	0.6874	0.6775	0.6969	0.0000	0.0039	0.0000
10	0.6756	0.6653	0.6845	0.0000	0.0037	0.0000
11	0.6687	0.6606	0.6776	0.0000	0.0036	0.0000
12	0.6658	0.6569	0.6756	0.0000	0.0037	0.0000
13	0.6634	0.6549	0.6719	0.0000	0.0035	0.0000
14	0.6598	0.6516	0.6677	0.0000	0.0033	0.0000
15	0.6552	0.6482	0.6616	0.0000	0.0031	0.0000
16	0.6499	0.6425	0.6563	0.0000	0.0030	0.0000
17	0.6453	0.6389	0.6516	0.0000	0.0029	0.0000
18	0.6414	0.6350	0.6472	0.0000	0.0028	0.0000
19	0.6386	0.6326	0.6451	0.0000	0.0028	0.0000
20	0.6368	0.6309	0.6429	0.0000	0.0027	0.0000
21	0.6355	0.6294	0.6416	0.0000	0.0027	0.0000
22	0.6343	0.6277	0.6402	0.0000	0.0026	0.0000
23	0.6330	0.6270	0.6383	0.0000	0.0026	0.0000
24	0.6319	0.6253	0.6372	0.0000	0.0025	0.0000
25	0.6307	0.6248	0.6358	0.0000	0.0024	0.0000

Table 18 Baseline1 Change - 3 Week Aircraft Entry Stagger

Run Name:	Baseline1					
Changes Made:	4 Week Stagger between aircraft entry at O-Level					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9954	0.9835	0.9977	0.0000	0.0027	0.0400
2	0.9561	0.9333	0.9727	0.0001	0.0074	0.0000
3	0.8866	0.8638	0.9078	0.0001	0.0092	0.0000
4	0.8116	0.7949	0.8312	0.0001	0.0075	0.0000
5	0.7766	0.7578	0.7905	0.0000	0.0065	0.0000
6	0.7566	0.7397	0.7693	0.0000	0.0056	0.0000
7	0.7369	0.7239	0.7504	0.0000	0.0049	0.0000
8	0.7122	0.7009	0.7200	0.0000	0.0040	0.0000
9	0.6933	0.6823	0.7016	0.0000	0.0038	0.0000
10	0.6802	0.6697	0.6886	0.0000	0.0036	0.0000
11	0.6723	0.6620	0.6799	0.0000	0.0036	0.0000
12	0.6684	0.6576	0.6772	0.0000	0.0035	0.0000
13	0.6661	0.6558	0.6725	0.0000	0.0032	0.0000
14	0.6625	0.6521	0.6699	0.0000	0.0032	0.0000
15	0.6578	0.6477	0.6641	0.0000	0.0030	0.0000
16	0.6526	0.6432	0.6594	0.0000	0.0028	0.0000
17	0.6478	0.6371	0.6542	0.0000	0.0027	0.0000
18	0.6436	0.6347	0.6496	0.0000	0.0027	0.0000
19	0.6405	0.6313	0.6467	0.0000	0.0026	0.0000
20	0.6384	0.6291	0.6445	0.0000	0.0027	0.0000
21	0.6370	0.6283	0.6436	0.0000	0.0026	0.0000
22	0.6358	0.6282	0.6431	0.0000	0.0026	0.0000
23	0.6346	0.6271	0.6416	0.0000	0.0025	0.0000
24	0.6333	0.6262	0.6401	0.0000	0.0024	0.0000
25	0.6320	0.6259	0.6385	0.0000	0.0024	0.0000

Table 19 Baseline1 Change - 4 Week Aircraft Entry Stagger

Run Name:	Baseline1					
Changes Made:	5 Week Stagger between aircraft entry at O-Level					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9964	0.9889	0.9981	0.0000	0.0018	0.2000
2	0.9677	0.9466	0.9849	0.0000	0.0067	0.0000
3	0.9059	0.8873	0.9294	0.0001	0.0075	0.0000
4	0.8339	0.8115	0.8476	0.0001	0.0071	0.0000
5	0.7900	0.7737	0.8031	0.0000	0.0063	0.0000
6	0.7664	0.7534	0.7768	0.0000	0.0051	0.0000
7	0.7465	0.7335	0.7584	0.0000	0.0046	0.0000
8	0.7219	0.7094	0.7332	0.0000	0.0044	0.0000
9	0.7013	0.6915	0.7102	0.0000	0.0039	0.0000
10	0.6868	0.6761	0.6956	0.0000	0.0040	0.0000
11	0.6775	0.6680	0.6862	0.0000	0.0037	0.0000
12	0.6729	0.6636	0.6808	0.0000	0.0038	0.0000
13	0.6700	0.6595	0.6769	0.0000	0.0036	0.0000
14	0.6663	0.6553	0.6730	0.0000	0.0034	0.0000
15	0.6618	0.6521	0.6685	0.0000	0.0033	0.0000
16	0.6563	0.6485	0.6636	0.0000	0.0030	0.0000
17	0.6512	0.6438	0.6584	0.0000	0.0029	0.0000
18	0.6468	0.6390	0.6533	0.0000	0.0028	0.0000
19	0.6434	0.6354	0.6502	0.0000	0.0027	0.0000
20	0.6410	0.6321	0.6482	0.0000	0.0027	0.0000
21	0.6394	0.6309	0.6458	0.0000	0.0027	0.0000
22	0.6381	0.6292	0.6444	0.0000	0.0025	0.0000
23	0.6367	0.6280	0.6431	0.0000	0.0024	0.0000
24	0.6353	0.6286	0.6418	0.0000	0.0023	0.0000
25	0.6340	0.6275	0.6398	0.0000	0.0023	0.0000

Table 20 Baseline1 Change - 5 Week Aircraft Entry Stagger

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX R. EXPERIMENT 2 (CANNIBALIZATION) OUTPUT

Year	Baseline1- Can.	Baseline1 - NO Can.
1	0.9909	0.9885
2	0.9235	0.8831
3	0.8463	0.7880
4	0.7832	0.7065
5	0.7603	0.6758
6	0.7426	0.6551
7	0.7214	0.6355
8	0.6986	0.6147
9	0.6824	0.5942
10	0.6720	0.5774
11	0.6662	0.5650
12	0.6638	0.5563
13	0.6613	0.5508
14	0.6573	0.5483
15	0.6525	0.5464
16	0.6475	0.5440
17	0.6431	0.5405
18	0.6396	0.5361
19	0.6371	0.5315
20	0.6354	0.5273
21	0.6342	0.5236
22	0.6332	0.5206
23	0.6321	0.5183
24	0.6309	0.5169
25	0.6297	0.5160

Table 21 Baseline1 Change - Cannibalization Summary of Mean A_0

Run Name:	Baseline1					
Changes Made:	No Cannibalization at the O-Level					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9885	0.9724	0.9970	0.0000	0.0048	0.0000
2	0.8831	0.8510	0.9111	0.0002	0.0127	0.0000
3	0.7880	0.7620	0.8103	0.0001	0.0087	0.0000
4	0.7065	0.6831	0.7248	0.0001	0.0078	0.0000
5	0.6758	0.6584	0.6915	0.0000	0.0065	0.0000
6	0.6551	0.6356	0.6685	0.0000	0.0065	0.0000
7	0.6355	0.6196	0.6500	0.0000	0.0061	0.0000
8	0.6147	0.6002	0.6259	0.0000	0.0050	0.0000
9	0.5942	0.5818	0.6060	0.0000	0.0048	0.0000
10	0.5774	0.5671	0.5902	0.0000	0.0047	0.0000
11	0.5650	0.5539	0.5764	0.0000	0.0046	0.0000
12	0.5563	0.5424	0.5692	0.0000	0.0048	0.0000
13	0.5508	0.5384	0.5626	0.0000	0.0047	0.0000
14	0.5483	0.5350	0.5601	0.0000	0.0049	0.0000
15	0.5464	0.5351	0.5592	0.0000	0.0046	0.0000
16	0.5440	0.5316	0.5564	0.0000	0.0045	0.0000
17	0.5405	0.5309	0.5524	0.0000	0.0042	0.0000
18	0.5361	0.5260	0.5467	0.0000	0.0041	0.0000
19	0.5315	0.5225	0.5410	0.0000	0.0038	0.0000
20	0.5273	0.5189	0.5369	0.0000	0.0037	0.0000
21	0.5236	0.5153	0.5337	0.0000	0.0037	0.0000
22	0.5206	0.5124	0.5300	0.0000	0.0037	0.0000
23	0.5183	0.5099	0.5271	0.0000	0.0037	0.0000
24	0.5169	0.5086	0.5262	0.0000	0.0038	0.0000
25	0.5160	0.5079	0.5247	0.0000	0.0038	0.0000

Table 22 Baseline1 Change - Without Cannibalization

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX S. EXPERIMENT 3 (IMPROVED RELIABILITY) OUTPUT

Year	Baseline1	TBF +250hrs	TBF +500hrs	TBF +750hrs	TBF +1000hrs	Infinite TBF
1	0.9909	0.9996	1.0000	1.0000	1.0000	1.0000
2	0.9235	0.9875	0.9990	0.9999	1.0000	1.0000
3	0.8463	0.9045	0.9350	0.9561	0.9789	0.9825
4	0.7832	0.8334	0.8676	0.8880	0.8859	0.8915
5	0.7603	0.8098	0.8403	0.8661	0.8800	0.9094
6	0.7426	0.7917	0.8219	0.8406	0.8448	0.8968
7	0.7214	0.7667	0.7960	0.8164	0.8208	0.8838
8	0.6986	0.7433	0.7742	0.7977	0.8070	0.8762
9	0.6824	0.7276	0.7597	0.7857	0.7970	0.8629
10	0.6720	0.7177	0.7506	0.7766	0.7902	0.8582
11	0.6662	0.7118	0.7440	0.7682	0.7823	0.8508
12	0.6638	0.7084	0.7385	0.7624	0.7765	0.8492
13	0.6613	0.7043	0.7334	0.7558	0.7690	0.8414
14	0.6573	0.6989	0.7266	0.7475	0.7603	0.8397
15	0.6525	0.6924	0.7197	0.7404	0.7534	0.8379
16	0.6475	0.6867	0.7139	0.7350	0.7485	0.8370
17	0.6431	0.6820	0.7097	0.7310	0.7448	0.8378
18	0.6396	0.6788	0.7066	0.7281	0.7422	0.8351
19	0.6371	0.6766	0.7046	0.7260	0.7402	0.8333
20	0.6354	0.6751	0.7029	0.7240	0.7380	0.8323
21	0.6342	0.6736	0.7011	0.7218	0.7358	0.8312
22	0.6332	0.6721	0.6992	0.7197	0.7339	0.8307
23	0.6321	0.6703	0.6975	0.7181	0.7323	0.8307
24	0.6309	0.6688	0.6960	0.7166	0.7308	0.8310
25	0.6297	0.6675	0.6949	0.7151	0.7296	0.8306

Table 23 Baseline1 Change - Improved Module Reliability Summary of Mean A_0

Run Name:	Baseline1					
Changes Made:	250 Hours Addition to Each Module TBF					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9996	0.9993	0.9997	0.0000	0.0001	0.9200
2	0.9875	0.9771	0.9929	0.0000	0.0028	0.0000
3	0.9045	0.8906	0.9188	0.0000	0.0057	0.0000
4	0.8334	0.8216	0.8461	0.0000	0.0053	0.0000
5	0.8098	0.7983	0.8232	0.0000	0.0052	0.0000
6	0.7917	0.7822	0.8018	0.0000	0.0045	0.0000
7	0.7667	0.7574	0.7754	0.0000	0.0040	0.0000
8	0.7433	0.7332	0.7518	0.0000	0.0038	0.0000
9	0.7276	0.7165	0.7362	0.0000	0.0036	0.0000
10	0.7177	0.7075	0.7266	0.0000	0.0038	0.0000
11	0.7118	0.7022	0.7203	0.0000	0.0036	0.0000
12	0.7084	0.7016	0.7165	0.0000	0.0031	0.0000
13	0.7043	0.6973	0.7106	0.0000	0.0028	0.0000
14	0.6989	0.6931	0.7055	0.0000	0.0027	0.0000
15	0.6924	0.6870	0.6974	0.0000	0.0025	0.0000
16	0.6867	0.6803	0.6928	0.0000	0.0024	0.0000
17	0.6820	0.6755	0.6875	0.0000	0.0026	0.0000
18	0.6788	0.6724	0.6851	0.0000	0.0025	0.0000
19	0.6766	0.6708	0.6822	0.0000	0.0025	0.0000
20	0.6751	0.6690	0.6807	0.0000	0.0025	0.0000
21	0.6736	0.6678	0.6787	0.0000	0.0024	0.0000
22	0.6721	0.6668	0.6777	0.0000	0.0023	0.0000
23	0.6703	0.6647	0.6758	0.0000	0.0022	0.0000
24	0.6688	0.6640	0.6738	0.0000	0.0021	0.0000
25	0.6675	0.6621	0.6724	0.0000	0.0021	0.0000

Table 24 Baseline1 Change - Module Time Between Failure + 250 Hours

Run Name:	Baseline1					
Changes Made:	500 Hours Addition to Each Module TBF					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	1.0000	1.0000	1.0000	0.0000	0.0000	1.0000
2	0.9990	0.9976	0.9995	0.0000	0.0004	0.0300
3	0.9350	0.9253	0.9446	0.0000	0.0038	0.0000
4	0.8676	0.8602	0.8766	0.0000	0.0036	0.0000
5	0.8403	0.8329	0.8517	0.0000	0.0037	0.0000
6	0.8219	0.8169	0.8299	0.0000	0.0030	0.0000
7	0.7960	0.7859	0.8036	0.0000	0.0033	0.0000
8	0.7742	0.7665	0.7815	0.0000	0.0032	0.0000
9	0.7597	0.7505	0.7678	0.0000	0.0031	0.0000
10	0.7506	0.7414	0.7576	0.0000	0.0031	0.0000
11	0.7440	0.7334	0.7501	0.0000	0.0029	0.0000
12	0.7385	0.7303	0.7444	0.0000	0.0029	0.0000
13	0.7334	0.7251	0.7387	0.0000	0.0028	0.0000
14	0.7266	0.7194	0.7327	0.0000	0.0027	0.0000
15	0.7197	0.7114	0.7260	0.0000	0.0027	0.0000
16	0.7139	0.7062	0.7198	0.0000	0.0026	0.0000
17	0.7097	0.7017	0.7160	0.0000	0.0027	0.0000
18	0.7066	0.6982	0.7128	0.0000	0.0026	0.0000
19	0.7046	0.6965	0.7109	0.0000	0.0025	0.0000
20	0.7029	0.6950	0.7084	0.0000	0.0024	0.0000
21	0.7011	0.6935	0.7067	0.0000	0.0024	0.0000
22	0.6992	0.6915	0.7054	0.0000	0.0024	0.0000
23	0.6975	0.6900	0.7031	0.0000	0.0025	0.0000
24	0.6960	0.6887	0.7024	0.0000	0.0025	0.0000
25	0.6949	0.6883	0.7011	0.0000	0.0024	0.0000

Table 25 Baseline1 Change - Module Time Between Failure + 500 Hours

Run Name:	Baseline1					
Changes Made:	750 Hours Addition to Each Module TBF					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	1.0000	1.0000	1.0000	0.0000	0.0000	1.0000
2	0.9999	0.9999	1.0000	0.0000	0.0000	1.0000
3	0.9561	0.9487	0.9641	0.0000	0.0032	0.0000
4	0.8880	0.8790	0.8956	0.0000	0.0031	0.0000
5	0.8661	0.8577	0.8763	0.0000	0.0035	0.0000
6	0.8406	0.8334	0.8493	0.0000	0.0035	0.0000
7	0.8164	0.8079	0.8239	0.0000	0.0032	0.0000
8	0.7977	0.7905	0.8042	0.0000	0.0031	0.0000
9	0.7857	0.7784	0.7940	0.0000	0.0031	0.0000
10	0.7766	0.7709	0.7824	0.0000	0.0027	0.0000
11	0.7682	0.7615	0.7737	0.0000	0.0025	0.0000
12	0.7624	0.7571	0.7684	0.0000	0.0025	0.0000
13	0.7558	0.7508	0.7610	0.0000	0.0023	0.0000
14	0.7475	0.7417	0.7521	0.0000	0.0023	0.0000
15	0.7404	0.7343	0.7454	0.0000	0.0024	0.0000
16	0.7350	0.7289	0.7399	0.0000	0.0023	0.0000
17	0.7310	0.7246	0.7368	0.0000	0.0023	0.0000
18	0.7281	0.7225	0.7336	0.0000	0.0022	0.0000
19	0.7260	0.7197	0.7312	0.0000	0.0022	0.0000
20	0.7240	0.7187	0.7294	0.0000	0.0022	0.0000
21	0.7218	0.7163	0.7273	0.0000	0.0021	0.0000
22	0.7197	0.7140	0.7244	0.0000	0.0021	0.0000
23	0.7181	0.7123	0.7230	0.0000	0.0021	0.0000
24	0.7166	0.7113	0.7226	0.0000	0.0020	0.0000
25	0.7151	0.7107	0.7201	0.0000	0.0020	0.0000

Table 26 Baseline1 Change - Module Time Between Failure + 750 Hours

Run Name:	Baseline1					
Changes Made:	1000 Hours Addition to Each Module TBF					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	1.0000	1.0000	1.0000	0.0000	0.0000	1.0000
2	1.0000	1.0000	1.0000	0.0000	0.0000	1.0000
3	0.9789	0.9767	0.9804	0.0000	0.0007	0.0000
4	0.8859	0.8789	0.8923	0.0000	0.0027	0.0000
5	0.8800	0.8738	0.8872	0.0000	0.0026	0.0000
6	0.8448	0.8349	0.8534	0.0000	0.0031	0.0000
7	0.8208	0.8140	0.8310	0.0000	0.0029	0.0000
8	0.8070	0.7986	0.8169	0.0000	0.0031	0.0000
9	0.7970	0.7905	0.8067	0.0000	0.0029	0.0000
10	0.7902	0.7841	0.8019	0.0000	0.0028	0.0000
11	0.7823	0.7749	0.7925	0.0000	0.0027	0.0000
12	0.7765	0.7697	0.7847	0.0000	0.0027	0.0000
13	0.7690	0.7623	0.7774	0.0000	0.0026	0.0000
14	0.7603	0.7545	0.7664	0.0000	0.0024	0.0000
15	0.7534	0.7480	0.7598	0.0000	0.0025	0.0000
16	0.7485	0.7431	0.7555	0.0000	0.0025	0.0000
17	0.7448	0.7393	0.7510	0.0000	0.0023	0.0000
18	0.7422	0.7370	0.7492	0.0000	0.0023	0.0000
19	0.7402	0.7346	0.7458	0.0000	0.0024	0.0000
20	0.7380	0.7328	0.7440	0.0000	0.0024	0.0000
21	0.7358	0.7304	0.7411	0.0000	0.0023	0.0000
22	0.7339	0.7287	0.7385	0.0000	0.0023	0.0000
23	0.7323	0.7274	0.7366	0.0000	0.0022	0.0000
24	0.7308	0.7258	0.7348	0.0000	0.0021	0.0000
25	0.7296	0.7245	0.7335	0.0000	0.0021	0.0000

Table 27 Baseline1 Change - Module Time Between Failure + 1000 Hours

Run Name:	Baseline1					
Changes Made:	9999999.9 (Infinite) Hours Addition to Each Module TBF					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	1.0000	1.0000	1.0000	0.0000	0.0000	1.0000
2	1.0000	1.0000	1.0000	0.0000	0.0000	1.0000
3	0.9825	0.9815	0.9833	0.0000	0.0004	0.0000
4	0.8915	0.8845	0.9015	0.0000	0.0029	0.0000
5	0.9094	0.9002	0.9185	0.0000	0.0030	0.0000
6	0.8968	0.8925	0.9013	0.0000	0.0019	0.0000
7	0.8838	0.8776	0.8894	0.0000	0.0024	0.0000
8	0.8762	0.8718	0.8800	0.0000	0.0018	0.0000
9	0.8629	0.8573	0.8677	0.0000	0.0021	0.0000
10	0.8582	0.8530	0.8622	0.0000	0.0018	0.0000
11	0.8508	0.8464	0.8557	0.0000	0.0019	0.0000
12	0.8492	0.8446	0.8543	0.0000	0.0017	0.0000
13	0.8414	0.8368	0.8462	0.0000	0.0018	0.0000
14	0.8397	0.8346	0.8454	0.0000	0.0019	0.0000
15	0.8379	0.8328	0.8436	0.0000	0.0019	0.0000
16	0.8370	0.8318	0.8437	0.0000	0.0019	0.0000
17	0.8378	0.8322	0.8440	0.0000	0.0019	0.0000
18	0.8351	0.8292	0.8412	0.0000	0.0019	0.0000
19	0.8333	0.8273	0.8390	0.0000	0.0019	0.0000
20	0.8323	0.8270	0.8383	0.0000	0.0019	0.0000
21	0.8312	0.8266	0.8369	0.0000	0.0019	0.0000
22	0.8307	0.8265	0.8354	0.0000	0.0018	0.0000
23	0.8307	0.8263	0.8355	0.0000	0.0017	0.0000
24	0.8310	0.8268	0.8356	0.0000	0.0018	0.0000
25	0.8306	0.8269	0.8351	0.0000	0.0017	0.0000

Table 28 Baseline1 Change - No Module Failure

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX T. EXPERIMENT 4 (MEAN D-LEVEL RTAT) OUTPUT

Year	Baseline1	200Day Mean	150Day Mean	100Day Mean	65Day Mean	50Day Mean	25Day Mean
1	0.9909	0.9877	0.9905	0.9941	0.9961	0.9967	0.9973
2	0.9235	0.9087	0.9379	0.9685	0.9861	0.9913	0.9957
3	0.8463	0.8535	0.8888	0.9330	0.9672	0.9799	0.9928
4	0.7832	0.8096	0.8586	0.9110	0.9528	0.9699	0.9900
5	0.7603	0.7837	0.8385	0.8977	0.9424	0.9614	0.9855
6	0.7426	0.7702	0.8207	0.8799	0.9276	0.9487	0.9794
7	0.7214	0.7512	0.8004	0.8611	0.9130	0.9376	0.9751
8	0.6986	0.7304	0.7819	0.8484	0.9071	0.9339	0.9742
9	0.6824	0.7141	0.7710	0.8434	0.9057	0.9339	0.9749
10	0.6720	0.7056	0.7661	0.8435	0.9057	0.9332	0.9736
11	0.6662	0.7011	0.7662	0.8423	0.9024	0.9291	0.9704
12	0.6638	0.7014	0.7648	0.8379	0.8973	0.9251	0.9690
13	0.6613	0.6999	0.7612	0.8333	0.8953	0.9238	0.9688
14	0.6573	0.6968	0.7567	0.8309	0.8934	0.9229	0.9689
15	0.6525	0.6930	0.7536	0.8284	0.8931	0.9229	0.9690
16	0.6475	0.6897	0.7507	0.8276	0.8933	0.9232	0.9693
17	0.6431	0.6868	0.7487	0.8279	0.8935	0.9234	0.9697
18	0.6396	0.6841	0.7484	0.8279	0.8938	0.9239	0.9699
19	0.6371	0.6829	0.7486	0.8280	0.8939	0.9237	0.9697
20	0.6354	0.6827	0.7485	0.8282	0.8932	0.9231	0.9697
21	0.6342	0.6827	0.7484	0.8276	0.8928	0.9232	0.9696
22	0.6332	0.6825	0.7483	0.8267	0.8929	0.9229	0.9691
23	0.6321	0.6824	0.7476	0.8265	0.8924	0.9222	0.9686
24	0.6309	0.6823	0.7466	0.8265	0.8915	0.9213	0.9684
25	0.6297	0.6815	0.7462	0.8259	0.8906	0.9207	0.9683

Table 29 Baseline1 Change - Mean D-Level RTAT Summary of Mean A_0

Run Name:	Baseline1					
Changes Made:	25 Day Mean RTAT (No RTAT Variability)					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9973	0.9961	0.9978	0.0000	0.0003	0.2200
2	0.9957	0.9939	0.9967	0.0000	0.0005	0.0000
3	0.9928	0.9913	0.9943	0.0000	0.0007	0.0000
4	0.9900	0.9879	0.9917	0.0000	0.0006	0.0000
5	0.9855	0.9839	0.9873	0.0000	0.0007	0.0000
6	0.9794	0.9777	0.9811	0.0000	0.0008	0.0000
7	0.9751	0.9729	0.9765	0.0000	0.0007	0.0000
8	0.9742	0.9724	0.9756	0.0000	0.0007	0.0000
9	0.9749	0.9735	0.9765	0.0000	0.0007	0.0000
10	0.9736	0.9722	0.9754	0.0000	0.0006	0.0000
11	0.9704	0.9692	0.9720	0.0000	0.0007	0.0000
12	0.9690	0.9677	0.9706	0.0000	0.0006	0.0000
13	0.9688	0.9671	0.9704	0.0000	0.0006	0.0000
14	0.9689	0.9677	0.9706	0.0000	0.0006	0.0000
15	0.9690	0.9675	0.9705	0.0000	0.0005	0.0000
16	0.9693	0.9679	0.9709	0.0000	0.0005	0.0000
17	0.9697	0.9683	0.9714	0.0000	0.0005	0.0000
18	0.9699	0.9686	0.9715	0.0000	0.0005	0.0000
19	0.9697	0.9684	0.9711	0.0000	0.0005	0.0000
20	0.9697	0.9681	0.9713	0.0000	0.0005	0.0000
21	0.9696	0.9681	0.9711	0.0000	0.0005	0.0000
22	0.9691	0.9678	0.9707	0.0000	0.0005	0.0000
23	0.9686	0.9673	0.9702	0.0000	0.0005	0.0000
24	0.9684	0.9672	0.9699	0.0000	0.0005	0.0000
25	0.9683	0.9671	0.9697	0.0000	0.0005	0.0000

Table 30 Baseline1 Change - 25 Day Mean D-Level RTAT

Run Name:	Baseline1					
Changes Made:	50 Day Mean RTAT (No RTAT Variability)					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9967	0.9930	0.9977	0.0000	0.0009	0.0000
2	0.9913	0.9880	0.9941	0.0000	0.0013	0.0000
3	0.9799	0.9755	0.9841	0.0000	0.0016	0.0000
4	0.9699	0.9662	0.9743	0.0000	0.0014	0.0000
5	0.9614	0.9589	0.9649	0.0000	0.0013	0.0000
6	0.9487	0.9454	0.9514	0.0000	0.0013	0.0000
7	0.9376	0.9345	0.9401	0.0000	0.0012	0.0000
8	0.9339	0.9309	0.9371	0.0000	0.0012	0.0000
9	0.9339	0.9306	0.9372	0.0000	0.0012	0.0000
10	0.9332	0.9303	0.9366	0.0000	0.0011	0.0000
11	0.9291	0.9265	0.9317	0.0000	0.0010	0.0000
12	0.9251	0.9233	0.9280	0.0000	0.0009	0.0000
13	0.9238	0.9221	0.9260	0.0000	0.0010	0.0000
14	0.9229	0.9202	0.9251	0.0000	0.0009	0.0000
15	0.9229	0.9207	0.9250	0.0000	0.0009	0.0000
16	0.9232	0.9211	0.9248	0.0000	0.0009	0.0000
17	0.9234	0.9211	0.9256	0.0000	0.0008	0.0000
18	0.9239	0.9214	0.9259	0.0000	0.0009	0.0000
19	0.9237	0.9214	0.9257	0.0000	0.0009	0.0000
20	0.9231	0.9206	0.9249	0.0000	0.0009	0.0000
21	0.9232	0.9207	0.9253	0.0000	0.0009	0.0000
22	0.9229	0.9207	0.9248	0.0000	0.0008	0.0000
23	0.9222	0.9203	0.9241	0.0000	0.0008	0.0000
24	0.9213	0.9196	0.9234	0.0000	0.0008	0.0000
25	0.9207	0.9188	0.9226	0.0000	0.0008	0.0000

Table 31 Baseline1 Change - 50 Day Mean D-Level RTAT

Run Name:	Baseline1					
Changes Made:	65 Day Mean RTAT (No RTAT Variability)					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9961	0.9925	0.9977	0.0000	0.0012	0.0100
2	0.9861	0.9807	0.9924	0.0000	0.0023	0.0000
3	0.9672	0.9589	0.9729	0.0000	0.0026	0.0000
4	0.9528	0.9484	0.9577	0.0000	0.0020	0.0000
5	0.9424	0.9373	0.9462	0.0000	0.0018	0.0000
6	0.9276	0.9227	0.9307	0.0000	0.0016	0.0000
7	0.9130	0.9093	0.9175	0.0000	0.0016	0.0000
8	0.9071	0.9035	0.9103	0.0000	0.0015	0.0000
9	0.9057	0.9021	0.9097	0.0000	0.0015	0.0000
10	0.9057	0.9025	0.9093	0.0000	0.0014	0.0000
11	0.9024	0.8990	0.9054	0.0000	0.0012	0.0000
12	0.8973	0.8934	0.9000	0.0000	0.0013	0.0000
13	0.8953	0.8919	0.8978	0.0000	0.0012	0.0000
14	0.8934	0.8909	0.8959	0.0000	0.0011	0.0000
15	0.8931	0.8905	0.8958	0.0000	0.0011	0.0000
16	0.8933	0.8902	0.8958	0.0000	0.0011	0.0000
17	0.8935	0.8906	0.8963	0.0000	0.0011	0.0000
18	0.8938	0.8911	0.8966	0.0000	0.0011	0.0000
19	0.8939	0.8911	0.8963	0.0000	0.0011	0.0000
20	0.8932	0.8910	0.8957	0.0000	0.0010	0.0000
21	0.8928	0.8907	0.8953	0.0000	0.0010	0.0000
22	0.8929	0.8905	0.8953	0.0000	0.0010	0.0000
23	0.8924	0.8901	0.8950	0.0000	0.0009	0.0000
24	0.8915	0.8895	0.8940	0.0000	0.0009	0.0000
25	0.8906	0.8885	0.8928	0.0000	0.0009	0.0000

Table 32 Baseline1 Change - 65 Day Mean D-Level RTAT

Run Name:	Baseline1					
Changes Made:	100 Day Mean RTAT (No RTAT Variability)					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9941	0.9867	0.9974	0.0000	0.0021	0.0000
2	0.9685	0.9582	0.9786	0.0000	0.0040	0.0000
3	0.9330	0.9216	0.9421	0.0000	0.0036	0.0000
4	0.9110	0.9050	0.9172	0.0000	0.0026	0.0000
5	0.8977	0.8920	0.9024	0.0000	0.0022	0.0000
6	0.8799	0.8751	0.8842	0.0000	0.0019	0.0000
7	0.8611	0.8545	0.8646	0.0000	0.0018	0.0000
8	0.8484	0.8425	0.8517	0.0000	0.0019	0.0000
9	0.8434	0.8378	0.8479	0.0000	0.0018	0.0000
10	0.8435	0.8381	0.8469	0.0000	0.0016	0.0000
11	0.8423	0.8373	0.8458	0.0000	0.0015	0.0000
12	0.8379	0.8327	0.8408	0.0000	0.0014	0.0000
13	0.8333	0.8290	0.8374	0.0000	0.0014	0.0000
14	0.8309	0.8277	0.8332	0.0000	0.0012	0.0000
15	0.8284	0.8246	0.8314	0.0000	0.0012	0.0000
16	0.8276	0.8234	0.8302	0.0000	0.0013	0.0000
17	0.8279	0.8237	0.8305	0.0000	0.0013	0.0000
18	0.8279	0.8237	0.8306	0.0000	0.0013	0.0000
19	0.8280	0.8239	0.8307	0.0000	0.0012	0.0000
20	0.8282	0.8242	0.8305	0.0000	0.0012	0.0000
21	0.8276	0.8239	0.8298	0.0000	0.0012	0.0000
22	0.8267	0.8228	0.8292	0.0000	0.0012	0.0000
23	0.8265	0.8233	0.8293	0.0000	0.0011	0.0000
24	0.8265	0.8231	0.8296	0.0000	0.0010	0.0000
25	0.8259	0.8225	0.8288	0.0000	0.0010	0.0000

Table 33 Baseline1 Change - 100 Day Mean D-Level RTAT

Run Name:	Baseline1					
Changes Made:	150 Day Mean RTAT (No RTAT Variability)					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9905	0.9809	0.9963	0.0000	0.0031	0.0000
2	0.9379	0.9247	0.9511	0.0000	0.0064	0.0000
3	0.8888	0.8745	0.9028	0.0000	0.0053	0.0000
4	0.8586	0.8437	0.8657	0.0000	0.0041	0.0000
5	0.8385	0.8271	0.8479	0.0000	0.0035	0.0000
6	0.8207	0.8118	0.8284	0.0000	0.0028	0.0000
7	0.8004	0.7928	0.8063	0.0000	0.0027	0.0000
8	0.7819	0.7740	0.7868	0.0000	0.0024	0.0000
9	0.7710	0.7642	0.7769	0.0000	0.0025	0.0000
10	0.7661	0.7591	0.7716	0.0000	0.0022	0.0000
11	0.7662	0.7606	0.7707	0.0000	0.0021	0.0000
12	0.7648	0.7606	0.7694	0.0000	0.0020	0.0000
13	0.7612	0.7571	0.7656	0.0000	0.0018	0.0000
14	0.7567	0.7532	0.7614	0.0000	0.0017	0.0000
15	0.7536	0.7493	0.7575	0.0000	0.0017	0.0000
16	0.7507	0.7463	0.7552	0.0000	0.0017	0.0000
17	0.7487	0.7435	0.7530	0.0000	0.0016	0.0000
18	0.7484	0.7433	0.7521	0.0000	0.0016	0.0000
19	0.7486	0.7443	0.7521	0.0000	0.0016	0.0000
20	0.7485	0.7446	0.7516	0.0000	0.0015	0.0000
21	0.7484	0.7447	0.7519	0.0000	0.0015	0.0000
22	0.7483	0.7444	0.7516	0.0000	0.0014	0.0000
23	0.7476	0.7443	0.7510	0.0000	0.0014	0.0000
24	0.7466	0.7437	0.7495	0.0000	0.0012	0.0000
25	0.7462	0.7429	0.7487	0.0000	0.0013	0.0000

Table 34 Baseline1 Change - 150 Day Mean D-Level RTAT

Run Name:	Baseline1					
Changes Made:	200 Day Mean RTAT (No RTAT Variability)					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9877	0.9777	0.9961	0.0000	0.0042	0.0000
2	0.9087	0.8914	0.9277	0.0001	0.0078	0.0000
3	0.8535	0.8402	0.8669	0.0000	0.0053	0.0000
4	0.8096	0.7991	0.8201	0.0000	0.0041	0.0000
5	0.7837	0.7757	0.7930	0.0000	0.0038	0.0000
6	0.7702	0.7632	0.7783	0.0000	0.0030	0.0000
7	0.7512	0.7453	0.7565	0.0000	0.0026	0.0000
8	0.7304	0.7246	0.7367	0.0000	0.0022	0.0000
9	0.7141	0.7076	0.7185	0.0000	0.0022	0.0000
10	0.7056	0.7006	0.7102	0.0000	0.0021	0.0000
11	0.7011	0.6956	0.7059	0.0000	0.0021	0.0000
12	0.7014	0.6953	0.7065	0.0000	0.0022	0.0000
13	0.6999	0.6954	0.7038	0.0000	0.0020	0.0000
14	0.6968	0.6925	0.7015	0.0000	0.0020	0.0000
15	0.6930	0.6891	0.6971	0.0000	0.0019	0.0000
16	0.6897	0.6854	0.6931	0.0000	0.0017	0.0000
17	0.6868	0.6828	0.6902	0.0000	0.0017	0.0000
18	0.6841	0.6799	0.6873	0.0000	0.0017	0.0000
19	0.6829	0.6789	0.6859	0.0000	0.0016	0.0000
20	0.6827	0.6790	0.6862	0.0000	0.0016	0.0000
21	0.6827	0.6790	0.6858	0.0000	0.0015	0.0000
22	0.6825	0.6788	0.6860	0.0000	0.0015	0.0000
23	0.6824	0.6777	0.6859	0.0000	0.0015	0.0000
24	0.6823	0.6782	0.6851	0.0000	0.0015	0.0000
25	0.6815	0.6777	0.6844	0.0000	0.0014	0.0000

Table 35 Baseline1 Change - 200 Day Mean D-Level RTAT

APPENDIX U. EXPERIMENT 5 (D-LVL RTAT COMPONENT) OUTPUT

Year	Baseline1	No Other(OT)	No OT/AWP	In Work Only
1	0.9909	0.9930	0.9935	0.9935
2	0.9235	0.9526	0.9570	0.9578
3	0.8463	0.9093	0.9158	0.9185
4	0.7832	0.8812	0.8904	0.8939
5	0.7603	0.8641	0.8747	0.8781
6	0.7426	0.8426	0.8548	0.8581
7	0.7214	0.8228	0.8358	0.8398
8	0.6986	0.8064	0.8209	0.8253
9	0.6824	0.7980	0.8136	0.8185
10	0.6720	0.7917	0.8094	0.8150
11	0.6662	0.7867	0.8068	0.8128
12	0.6638	0.7808	0.8016	0.8078
13	0.6613	0.7725	0.7941	0.8008
14	0.6573	0.7642	0.7881	0.7956
15	0.6525	0.7588	0.7848	0.7928
16	0.6475	0.7554	0.7826	0.7910
17	0.6431	0.7536	0.7817	0.7903
18	0.6396	0.7526	0.7812	0.7899
19	0.6371	0.7513	0.7799	0.7888
20	0.6354	0.7495	0.7791	0.7882
21	0.6342	0.7485	0.7793	0.7885
22	0.6332	0.7489	0.7793	0.7883
23	0.6321	0.7493	0.7793	0.7880
24	0.6309	0.7496	0.7794	0.7881
25	0.6297	0.7498	0.7792	0.7878

Table 36 Baseline1 Change - D-Level RTAT Component Summary of Mean A_0

Run Name:	Baseline1					
Changes Made:	D-Level RTAT Without Other Times					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9930	0.9833	0.9971	0.0000	0.0026	0.0000
2	0.9526	0.9377	0.9670	0.0000	0.0060	0.0000
3	0.9093	0.8956	0.9195	0.0000	0.0045	0.0000
4	0.8812	0.8689	0.8914	0.0000	0.0040	0.0000
5	0.8641	0.8570	0.8709	0.0000	0.0031	0.0000
6	0.8426	0.8346	0.8495	0.0000	0.0029	0.0000
7	0.8228	0.8160	0.8311	0.0000	0.0027	0.0000
8	0.8064	0.7995	0.8162	0.0000	0.0028	0.0000
9	0.7980	0.7930	0.8064	0.0000	0.0025	0.0000
10	0.7917	0.7865	0.8009	0.0000	0.0027	0.0000
11	0.7867	0.7783	0.7962	0.0000	0.0027	0.0000
12	0.7808	0.7725	0.7886	0.0000	0.0023	0.0000
13	0.7725	0.7664	0.7790	0.0000	0.0023	0.0000
14	0.7642	0.7567	0.7717	0.0000	0.0024	0.0000
15	0.7588	0.7512	0.7657	0.0000	0.0024	0.0000
16	0.7554	0.7480	0.7616	0.0000	0.0024	0.0000
17	0.7536	0.7459	0.7584	0.0000	0.0023	0.0000
18	0.7526	0.7461	0.7575	0.0000	0.0021	0.0000
19	0.7513	0.7446	0.7569	0.0000	0.0021	0.0000
20	0.7495	0.7430	0.7547	0.0000	0.0022	0.0000
21	0.7485	0.7423	0.7536	0.0000	0.0021	0.0000
22	0.7489	0.7424	0.7536	0.0000	0.0022	0.0000
23	0.7493	0.7427	0.7541	0.0000	0.0021	0.0000
24	0.7496	0.7440	0.7544	0.0000	0.0020	0.0000
25	0.7498	0.7446	0.7546	0.0000	0.0020	0.0000

Table 37 Baseline1 Change - D-Level RTAT Without "Other" Time

Run Name:	Baseline1					
Changes Made:	D-Level RTAT Without Other and AWP Times					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9935	0.9879	0.9975	0.0000	0.0024	0.0000
2	0.9570	0.9440	0.9685	0.0000	0.0053	0.0000
3	0.9158	0.9068	0.9268	0.0000	0.0047	0.0000
4	0.8904	0.8816	0.8995	0.0000	0.0038	0.0000
5	0.8747	0.8671	0.8819	0.0000	0.0033	0.0000
6	0.8548	0.8477	0.8612	0.0000	0.0029	0.0000
7	0.8358	0.8288	0.8428	0.0000	0.0028	0.0000
8	0.8209	0.8137	0.8266	0.0000	0.0026	0.0000
9	0.8136	0.8069	0.8195	0.0000	0.0026	0.0000
10	0.8094	0.8028	0.8151	0.0000	0.0023	0.0000
11	0.8068	0.7997	0.8115	0.0000	0.0023	0.0000
12	0.8016	0.7947	0.8064	0.0000	0.0022	0.0000
13	0.7941	0.7878	0.7984	0.0000	0.0021	0.0000
14	0.7881	0.7815	0.7930	0.0000	0.0022	0.0000
15	0.7848	0.7776	0.7908	0.0000	0.0022	0.0000
16	0.7826	0.7768	0.7878	0.0000	0.0021	0.0000
17	0.7817	0.7753	0.7871	0.0000	0.0020	0.0000
18	0.7812	0.7751	0.7863	0.0000	0.0020	0.0000
19	0.7799	0.7748	0.7842	0.0000	0.0020	0.0000
20	0.7791	0.7743	0.7833	0.0000	0.0019	0.0000
21	0.7793	0.7746	0.7832	0.0000	0.0019	0.0000
22	0.7793	0.7747	0.7834	0.0000	0.0018	0.0000
23	0.7793	0.7742	0.7837	0.0000	0.0017	0.0000
24	0.7794	0.7741	0.7833	0.0000	0.0017	0.0000
25	0.7792	0.7740	0.7832	0.0000	0.0016	0.0000

Table 38 Baseline1 Change - D-Level RTAT Without “Other” and AWP Time

Run Name:	Baseline1					
Changes Made:	D-Level RTAT With Only In Work Times					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9935	0.9844	0.9971	0.0000	0.0024	0.0000
2	0.9578	0.9452	0.9732	0.0000	0.0055	0.0000
3	0.9185	0.9081	0.9296	0.0000	0.0044	0.0000
4	0.8939	0.8852	0.9022	0.0000	0.0035	0.0000
5	0.8781	0.8682	0.8870	0.0000	0.0033	0.0000
6	0.8581	0.8498	0.8666	0.0000	0.0029	0.0000
7	0.8398	0.8331	0.8460	0.0000	0.0027	0.0000
8	0.8253	0.8194	0.8315	0.0000	0.0026	0.0000
9	0.8185	0.8127	0.8236	0.0000	0.0025	0.0000
10	0.8150	0.8084	0.8213	0.0000	0.0026	0.0000
11	0.8128	0.8062	0.8184	0.0000	0.0024	0.0000
12	0.8078	0.8030	0.8124	0.0000	0.0021	0.0000
13	0.8008	0.7957	0.8054	0.0000	0.0020	0.0000
14	0.7956	0.7910	0.8009	0.0000	0.0020	0.0000
15	0.7928	0.7864	0.7978	0.0000	0.0019	0.0000
16	0.7910	0.7844	0.7956	0.0000	0.0019	0.0000
17	0.7903	0.7842	0.7951	0.0000	0.0019	0.0000
18	0.7899	0.7843	0.7941	0.0000	0.0019	0.0000
19	0.7888	0.7826	0.7926	0.0000	0.0019	0.0000
20	0.7882	0.7823	0.7923	0.0000	0.0019	0.0000
21	0.7885	0.7834	0.7925	0.0000	0.0019	0.0000
22	0.7883	0.7835	0.7922	0.0000	0.0018	0.0000
23	0.7880	0.7836	0.7916	0.0000	0.0018	0.0000
24	0.7881	0.7834	0.7915	0.0000	0.0017	0.0000
25	0.7878	0.7835	0.7914	0.0000	0.0017	0.0000

Table 39 Baseline1 Change - D-Level RTAT With Only “In Work” Time

APPENDIX V. EXPERIMENT 6 (BUILD WINDOW) OUTPUT

Year	1000hr BW	750hr BW	500hr BW	250hr BW	50hr BW
1	0.9824	0.9913	0.9909	0.9910	0.9912
2	0.7692	0.8995	0.9235	0.9227	0.9218
3	0.6826	0.7817	0.8463	0.8681	0.8668
4	0.6434	0.7506	0.7832	0.8146	0.8318
5	0.6237	0.7281	0.7603	0.7680	0.7584
6	0.6083	0.7070	0.7426	0.7417	0.7367
7	0.5947	0.6874	0.7214	0.7291	0.7139
8	0.5845	0.6712	0.6986	0.7160	0.7010
9	0.5763	0.6604	0.6824	0.6940	0.6840
10	0.5692	0.6546	0.6720	0.6782	0.6608
11	0.5630	0.6505	0.6662	0.6713	0.6373
12	0.5579	0.6463	0.6638	0.6698	0.6232
13	0.5535	0.6415	0.6613	0.6644	0.6229
14	0.5496	0.6365	0.6573	0.6603	0.6160
15	0.5465	0.6316	0.6525	0.6591	0.6060
16	0.5437	0.6276	0.6475	0.6566	0.6045
17	0.5413	0.6242	0.6431	0.6511	0.6018
18	0.5392	0.6215	0.6396	0.6455	0.5998
19	0.5373	0.6193	0.6371	0.6408	0.5925
20	0.5356	0.6174	0.6354	0.6378	0.5857
21	0.5341	0.6157	0.6342	0.6360	0.5809
22	0.5327	0.6141	0.6332	0.6351	0.5766
23	0.5314	0.6126	0.6321	0.6344	0.5740
24	0.5303	0.6112	0.6309	0.6335	0.5706
25	0.5294	0.6101	0.6297	0.6325	0.5688

Table 40 Baseline1 Change – Build Window Summary of Mean A_0

Run Name:	Baseline1					
Changes Made:	50 hour Build Window					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9912	0.9823	0.9963	0.0000	0.0028	0.0000
2	0.9218	0.9043	0.9442	0.0001	0.0074	0.0000
3	0.8668	0.8427	0.8850	0.0001	0.0084	0.0000
4	0.8318	0.8029	0.8515	0.0001	0.0085	0.0000
5	0.7584	0.7401	0.7725	0.0000	0.0061	0.0000
6	0.7367	0.7221	0.7506	0.0000	0.0057	0.0000
7	0.7139	0.7014	0.7263	0.0000	0.0060	0.0000
8	0.7010	0.6882	0.7103	0.0000	0.0045	0.0000
9	0.6840	0.6720	0.6918	0.0000	0.0036	0.0000
10	0.6608	0.6522	0.6673	0.0000	0.0028	0.0000
11	0.6373	0.6303	0.6432	0.0000	0.0024	0.0000
12	0.6232	0.6142	0.6316	0.0000	0.0031	0.0000
13	0.6229	0.6136	0.6331	0.0000	0.0033	0.0000
14	0.6160	0.6108	0.6235	0.0000	0.0023	0.0000
15	0.6060	0.5993	0.6161	0.0000	0.0028	0.0000
16	0.6045	0.5973	0.6156	0.0000	0.0030	0.0000
17	0.6018	0.5952	0.6129	0.0000	0.0029	0.0000
18	0.5998	0.5952	0.6070	0.0000	0.0023	0.0000
19	0.5925	0.5884	0.5991	0.0000	0.0022	0.0000
20	0.5857	0.5802	0.5926	0.0000	0.0026	0.0000
21	0.5809	0.5750	0.5882	0.0000	0.0025	0.0000
22	0.5766	0.5697	0.5831	0.0000	0.0029	0.0000
23	0.5740	0.5678	0.5795	0.0000	0.0024	0.0000
24	0.5706	0.5647	0.5769	0.0000	0.0024	0.0000
25	0.5688	0.5630	0.5753	0.0000	0.0026	0.0000

Table 41 Baseline1 Change – 50 hour Build Window

Run Name:	Baseline1					
Changes Made:	250 hour Build Window					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9910	0.9818	0.9970	0.0000	0.0033	0.0000
2	0.9227	0.9025	0.9425	0.0001	0.0081	0.0000
3	0.8681	0.8414	0.8870	0.0001	0.0087	0.0000
4	0.8146	0.7978	0.8231	0.0000	0.0051	0.0000
5	0.7680	0.7468	0.7809	0.0000	0.0063	0.0000
6	0.7417	0.7273	0.7542	0.0000	0.0047	0.0000
7	0.7291	0.7183	0.7372	0.0000	0.0042	0.0000
8	0.7160	0.7076	0.7230	0.0000	0.0037	0.0000
9	0.6940	0.6872	0.7008	0.0000	0.0029	0.0000
10	0.6782	0.6714	0.6857	0.0000	0.0030	0.0000
11	0.6713	0.6638	0.6797	0.0000	0.0033	0.0000
12	0.6698	0.6622	0.6779	0.0000	0.0032	0.0000
13	0.6644	0.6583	0.6723	0.0000	0.0027	0.0000
14	0.6603	0.6534	0.6691	0.0000	0.0031	0.0000
15	0.6591	0.6518	0.6672	0.0000	0.0030	0.0000
16	0.6566	0.6507	0.6647	0.0000	0.0027	0.0000
17	0.6511	0.6453	0.6583	0.0000	0.0026	0.0000
18	0.6455	0.6386	0.6525	0.0000	0.0025	0.0000
19	0.6408	0.6349	0.6489	0.0000	0.0026	0.0000
20	0.6378	0.6319	0.6457	0.0000	0.0027	0.0000
21	0.6360	0.6299	0.6441	0.0000	0.0027	0.0000
22	0.6351	0.6292	0.6428	0.0000	0.0027	0.0000
23	0.6344	0.6291	0.6414	0.0000	0.0026	0.0000
24	0.6335	0.6280	0.6394	0.0000	0.0025	0.0000
25	0.6325	0.6269	0.6396	0.0000	0.0024	0.0000

Table 42 Baseline1 Change – 250 hour Build Window

Run Name:	Baseline1					
Changes Made:	750 hour Build Window					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9913	0.9841	0.9959	0.0000	0.0026	0.0000
2	0.8995	0.8837	0.9161	0.0000	0.0070	0.0000
3	0.7817	0.7676	0.7943	0.0000	0.0063	0.0000
4	0.7506	0.7387	0.7651	0.0000	0.0053	0.0000
5	0.7281	0.7171	0.7385	0.0000	0.0046	0.0000
6	0.7070	0.6948	0.7152	0.0000	0.0042	0.0000
7	0.6874	0.6763	0.6947	0.0000	0.0039	0.0000
8	0.6712	0.6618	0.6791	0.0000	0.0035	0.0000
9	0.6604	0.6526	0.6680	0.0000	0.0034	0.0000
10	0.6546	0.6460	0.6624	0.0000	0.0036	0.0000
11	0.6505	0.6419	0.6572	0.0000	0.0032	0.0000
12	0.6463	0.6387	0.6533	0.0000	0.0029	0.0000
13	0.6415	0.6344	0.6475	0.0000	0.0026	0.0000
14	0.6365	0.6303	0.6425	0.0000	0.0025	0.0000
15	0.6316	0.6267	0.6380	0.0000	0.0023	0.0000
16	0.6276	0.6220	0.6333	0.0000	0.0025	0.0000
17	0.6242	0.6185	0.6289	0.0000	0.0024	0.0000
18	0.6215	0.6156	0.6271	0.0000	0.0026	0.0000
19	0.6193	0.6140	0.6250	0.0000	0.0025	0.0000
20	0.6174	0.6118	0.6230	0.0000	0.0025	0.0000
21	0.6157	0.6098	0.6214	0.0000	0.0025	0.0000
22	0.6141	0.6093	0.6193	0.0000	0.0024	0.0000
23	0.6126	0.6077	0.6181	0.0000	0.0024	0.0000
24	0.6112	0.6072	0.6162	0.0000	0.0023	0.0000
25	0.6101	0.6050	0.6156	0.0000	0.0023	0.0000

Table 43 Baseline1 Change – 750 hour Build Window

Run Name:	Baseline1					
Changes Made:	1000 hour Build Window					
Number of Years:	25					
Runs Per Year:	100					
Year	Ao Mean	Ao Min	Ao Max	Ao Var	Ao St Dev	P(Spare)
1	0.9824	0.9724	0.9907	0.0000	0.0038	0.0000
2	0.7692	0.7421	0.7940	0.0001	0.0116	0.0000
3	0.6826	0.6575	0.7027	0.0001	0.0082	0.0000
4	0.6434	0.6238	0.6582	0.0001	0.0076	0.0000
5	0.6237	0.6087	0.6364	0.0000	0.0064	0.0000
6	0.6083	0.5916	0.6233	0.0000	0.0061	0.0000
7	0.5947	0.5809	0.6061	0.0000	0.0053	0.0000
8	0.5845	0.5727	0.5943	0.0000	0.0048	0.0000
9	0.5763	0.5661	0.5857	0.0000	0.0045	0.0000
10	0.5692	0.5606	0.5781	0.0000	0.0041	0.0000
11	0.5630	0.5544	0.5705	0.0000	0.0040	0.0000
12	0.5579	0.5498	0.5663	0.0000	0.0037	0.0000
13	0.5535	0.5458	0.5612	0.0000	0.0035	0.0000
14	0.5496	0.5412	0.5575	0.0000	0.0034	0.0000
15	0.5465	0.5391	0.5543	0.0000	0.0030	0.0000
16	0.5437	0.5368	0.5514	0.0000	0.0029	0.0000
17	0.5413	0.5330	0.5488	0.0000	0.0029	0.0000
18	0.5392	0.5320	0.5462	0.0000	0.0030	0.0000
19	0.5373	0.5292	0.5434	0.0000	0.0029	0.0000
20	0.5356	0.5281	0.5416	0.0000	0.0030	0.0000
21	0.5341	0.5265	0.5407	0.0000	0.0029	0.0000
22	0.5327	0.5249	0.5393	0.0000	0.0029	0.0000
23	0.5314	0.5229	0.5378	0.0000	0.0027	0.0000
24	0.5303	0.5232	0.5373	0.0000	0.0027	0.0000
25	0.5294	0.5223	0.5370	0.0000	0.0027	0.0000

Table 44 Baseline1 Change – 1000 hour Build Window

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Boeing, "F/A-18E/F Super Hornet" [http://www.boeing.com/defense-space/military/fa18ef/flash.html], 19 August 2003.

Burrows, J. and Gardner, J., *PC ARROWs Version 1.0 User's Manual, Revision 0, Change 0*, Navy Ships Parts Control Center (046), Mechanicsburg, Pennsylvania, March 1994.

Buss, A. "Simkit." [http://diana.gl.nps.navy.mil/Simkit/], 05 September 2003.

Buss, A., "Discrete Event Programming with Simkit," *Simulation News Europe*, Issue 32/33, November 2001.

Chief of Information, United States Navy Office of Information (CHINFO), "United States Navy Fact File" [http://www.chinfo.navy.mil/navpalib/factfile/ffiletop.html#air1], 24 August 2003.

Email between Gene F. Woodburn, Aircraft Engine Maintenance System Database Administrator , NAVAIR 3.6.3, Naval Air Systems Command, and the author. 11 June 2003.

Email between Gene F. Woodburn, Aircraft Engine Maintenance System Database Administrator, NAVAIR 3.6.3, Naval Air Systems Command, and the author. 13 August 2003.

Email between Major Richard Williamson, USMC, Code 03312, Naval Inventory Control Point Philadelphia, and the author. 09 December 2002.

Email between Major Richard Williamson, USMC, Code 03312, Naval Inventory Control Point Philadelphia, and the author. 30 April 2003.

Email between Major Richard Williamson, USMC, Code 03312, Naval Inventory Control Point Philadelphia, and the author. 11 August 2003.

Email between Major Richard Williamson, USMC, Code 03312, Naval Inventory Control Point Philadelphia, and the author. 13 August 2003.

Email between Major Richard Williamson, USMC, Code 03312, Naval Inventory Control Point Philadelphia, and the author. 25 August 2003.

Email between Major Richard Williamson, USMC, Code 03312, Naval Inventory Control Point Philadelphia, and the author. 26 August 2003.

General Electric, "Military Engines" [http://www.geae.com/engines/military/index.html], 25 August 2003.

Grossnick, R. A., *United States Naval Aviation,* 4th ed., Naval Historical Center, 1997.

Law, A. M. and Keaton, W. D., *Simulation Modeling and Analysis,* 3rd ed., McGraw-Hill Higher Education. 2000.

Office of the Chief of Naval Operations (OPNAV), Operational Availability Handbook, OPNAVINST 3000.12. 29 December 1987.

Office of the Deputy Secretary of Defense for Logistics and Materiel Readiness, Department of Defense (DoD), DoD Supply Chain Materiel Management Regulation, DoD 4140.1-R. 23 May 2003.

Savitch, W., *JAVA, An Introduction to Computer Science and Programming*, 2nd ed., Prentice Hall INC. 2001

Stearns III, D. E., *Logistics Simulation Metamodel for F404-GE-400 Engine Maintenance*, Master's Thesis. Naval Postgraduate School. Monterey, California. December 1998.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Major Richard Williamson, USMC, Code 03312
   Naval Inventory Control Point Philadelphia
   Philadelphia, Pennsylvania

4. Professor Arnold Buss
   Modeling, Virtual Environments and Simulation Institute
   Naval Postgraduate School
   Monterey, California

5. Commander Kevin Maher
   Naval Postgraduate School
   Monterey, California

6. Professor David Olwell
   Naval Postgraduate School
   Monterey, California

7. Lieutenant Commander Eric Schoch
   Quakertown, Pennsylvania